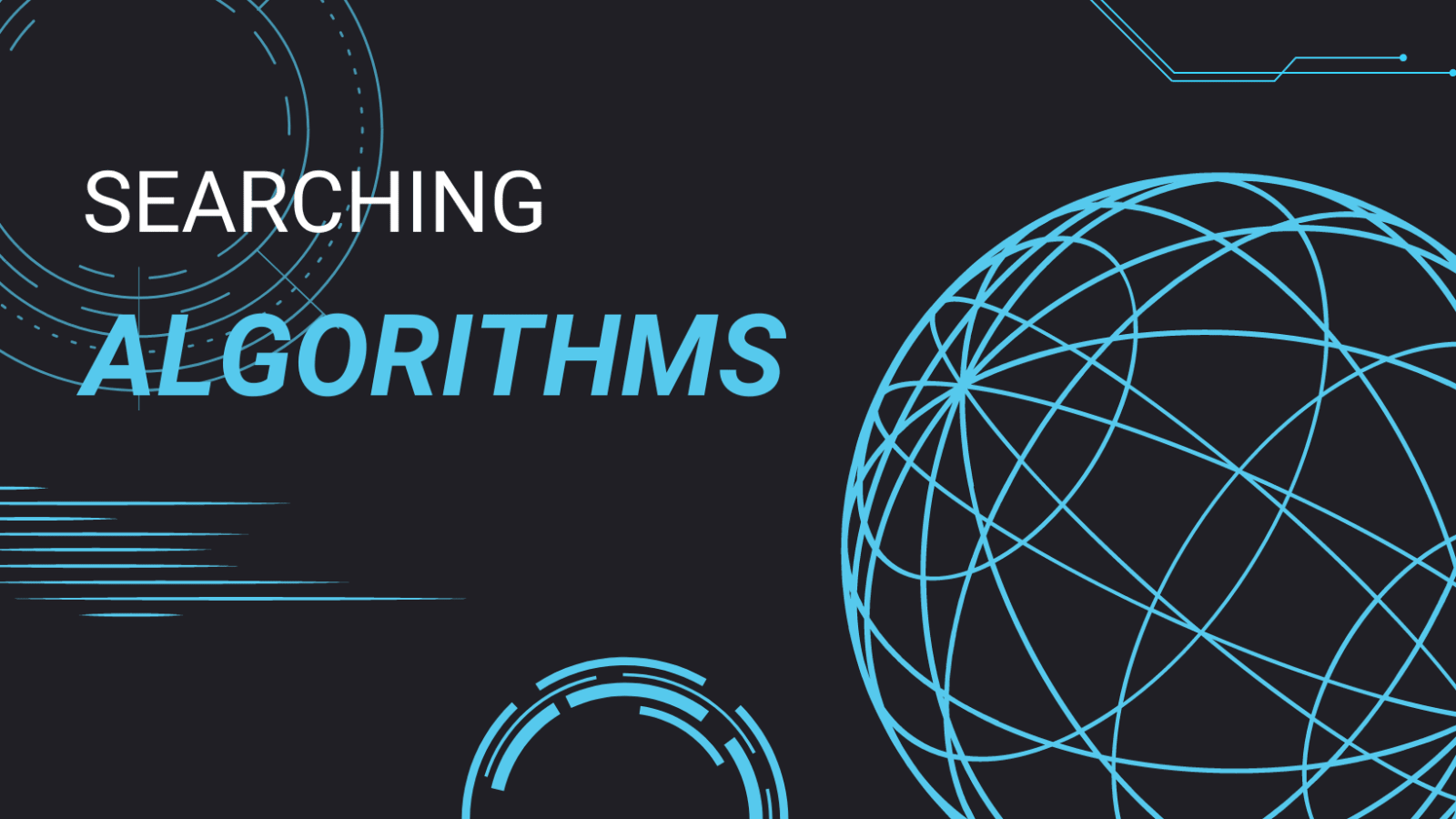
|  |  |  |  |
| --- | --- | --- | --- |
| Uninformed search algorithm |  |  | FOUNDATION OF AI WITH R LANGUAGE |
|  |  | XAI302/XAI305 |
|  |  | Bsc Artificial Intelligence II – Year |
| vikneshraj d |  |  | 122012173030 |
| • DEPTH FIRST SEARCH  • BREADTH FIRST SEARCH  • DEPTH LIMITED SEARCH  • ITERATIVE DEEPENING SEARCH  • BIDIRECTIONAL SEARCH  • UNIFORM COST SEARCH |  |  |  |
| [This Photo](https://www.picpedia.org/highway-signs/a/artificial-intelligence.html) by Unknown Author is licensed under [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/) |
|  |
|  |
|  |  |  |  |



**Uninformed search algorithm**

Uninformed search, also known as blind search, is a search algorithm that explores a problem space without any specific knowledge or information about the problem other than the initial state and the possible actions to take. It lacks domain-specific heuristics or prior knowledge about the problem.

Uninformed search algorithms, such as breadth-first search and depth-first search, systematically explore the search space by applying predefined rules to generate successor states until a goal state is found or the search is exhausted. These algorithms are typically less efficient than informed search algorithms but can be useful in certain scenarios or as a basis for more advanced search techniques.

Uninformed searches rely solely on the given problem definition and operate systematically to find a solution. Examples of uninformed search algorithms include breadth-first search ([BFS](https://www.educative.io/answers/what-is-breadth-first-search)), depth-first search ([DFS](https://www.educative.io/answers/what-is-depth-first-search)),uniform-cost search ([UCS](https://www.educative.io/answers/what-is-uniform-cost-search)), [depth-limited search](https://www.educative.io/answers/what-is-depth-limited-search) , and [iterative deepening depth-first search.](https://www.educative.io/answers/what-is-iterative-deepening-search) Although all these examples work in a brute force way, they differ in the way they traverse the nodes.

• DEPTH FIRST SEARCH

• BREADTH FIRST SEARCH

• DEPTH LIMITED SEARCH

• ITERATIVE DEEPENING SEARCH

• BIDIRECTIONAL SEARCH

• UNIFORM COST SEARCH

**DEPTH FIRST SEARCH**

**Depth-first search** (DFS) is a graph traversal algorithm that explores as far as possible along each branch before backtracking . It starts at the root node and explores as far as possible along each branch before backtracking . DFS can be implemented using a stack data structure . The algorithm is recursive in nature and can be used to search all the vertices of a tree data structure or a graph . Here is the step-by-step process to implement the DFS traversal :

1.Create a stack with the total number of vertices in the graph

2. Choose any vertex as the starting point of traversal, and push that vertex into the stack.

3. Push a non-visited vertex (adjacent to the vertex on the top of the stack) to the top of the stack.

4. Repeat steps 3 and 4 until no vertices are left to visit from the vertex on the stack’s top.

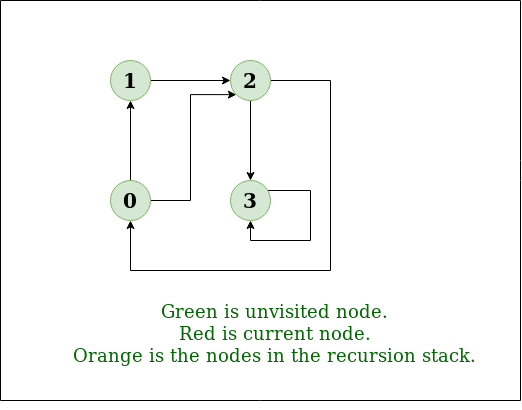
5. If no vertex is left, go back and pop a vertex from the stack.

6. Repeat steps 2, 3, and 4 until the stack is empty.

Overall, **DFS** is a versatile and widely used algorithm for exploring and traversing graphs and trees. Its choice over **BFS** or other algorithms depends on the specific problem requirements and the nature of the graph or tree being explored.

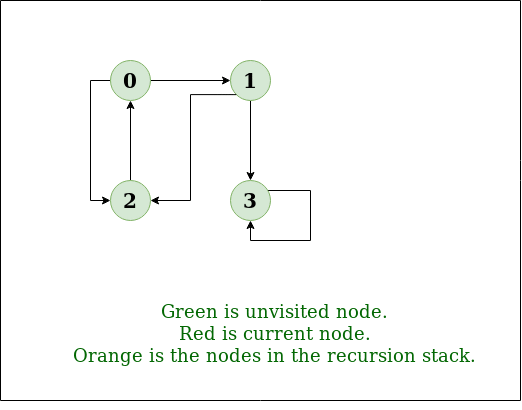
**Example:** 1

***Input:****n = 4, e = 6   
0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3****Output:****DFS from vertex 1 : 1 2 0 3****Explanation:****DFS Diagram:*



**Example:** 2

***Input:****n = 4, e = 6   
2 -> 0, 0 -> 2, 1 -> 2, 0 -> 1, 3 -> 3, 1 -> 3****Output:****DFS from vertex 2 : 2 0 1 3****Explanation:****DFS Diagram*



**BREADTH FIRST SEARCH**

**Breadth-first search** (BFS) is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using **BFS** for traversal, any node in the graph can be considered as the root node . Here are the steps involved in BFS algorithm to explore a graph:

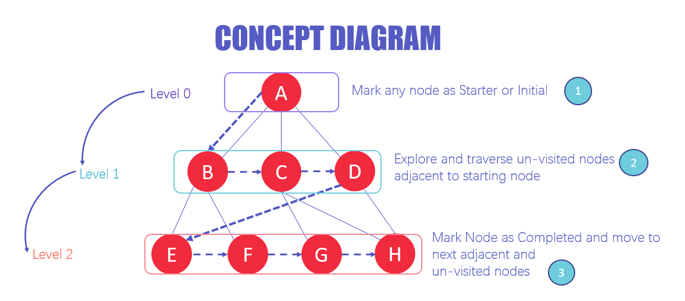
1. SET STATUS = 1 (ready state) for each node in G

2. Enqueue the starting node A and set its STATUS = 2 (waiting state)

3. Repeat Steps 4 and 5 until QUEUE is empty

4. Dequeue a node N. Process it and set its STATUS = 3 (processed state).

5. Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state) [END OF LOOP]

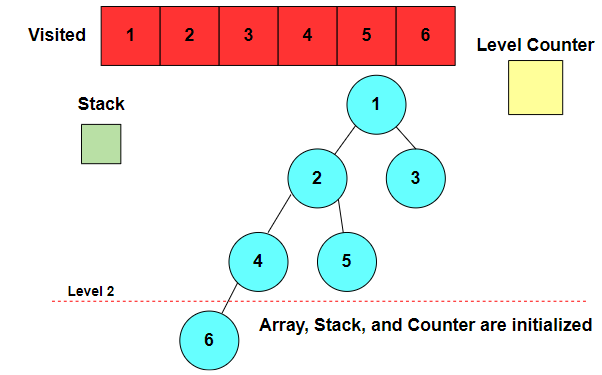


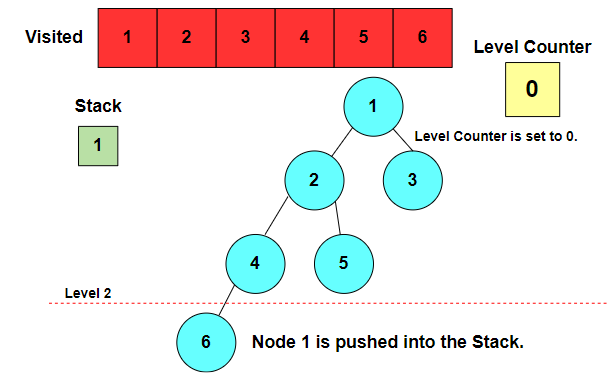
**DEPTH LIMITED SEARCH**

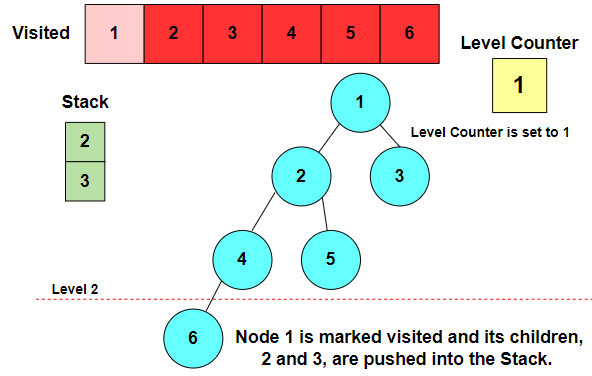
The **Depth-limited search** (DLS) method is almost equal to depth-first search (DFS), but **DLS** can work on the infinite state space problem because it bounds the depth of the search tree with a predetermined limit **L**. Nodes at this depth limit are treated as if they had no successors

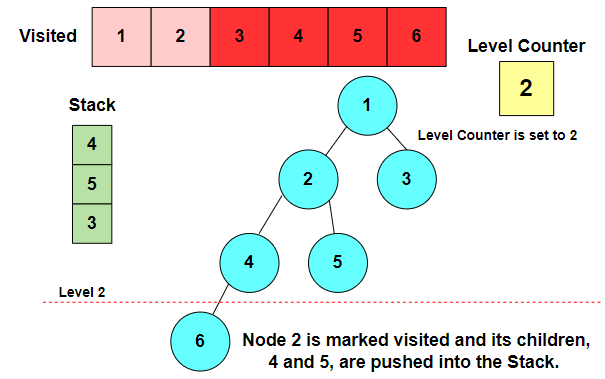
To implement DFS with a stack, we use these steps:

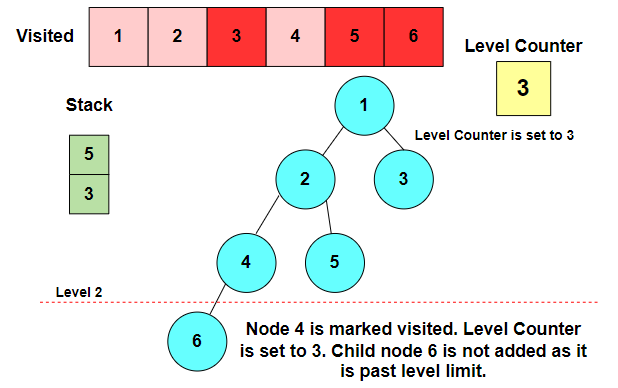
1. We push the root node into the stack.
2. We pop the root node before pushing all its child nodes into the stack.
3. We pop a child node before pushing all of its nodes back into the stack.
4. We repeat this process until we either find our result or traverse the whole tree.

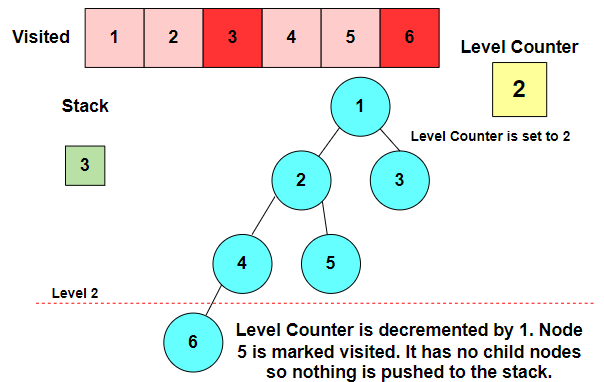


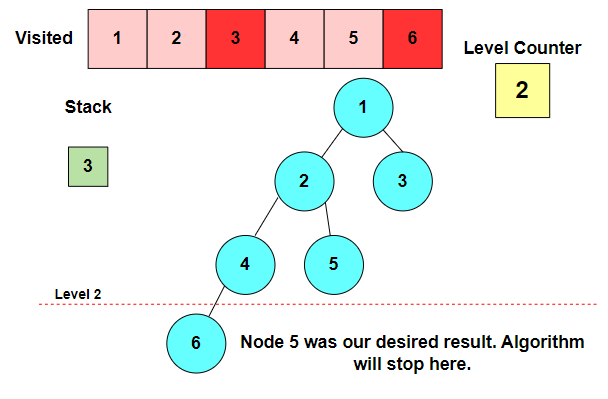












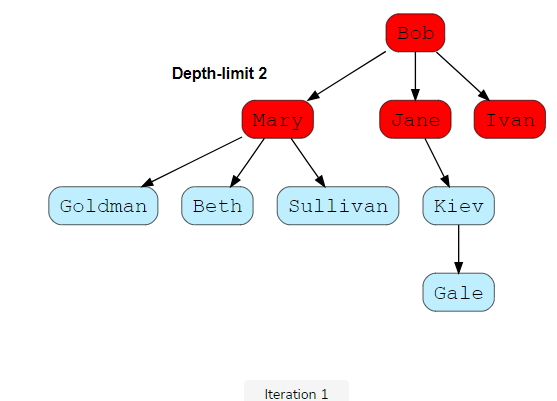
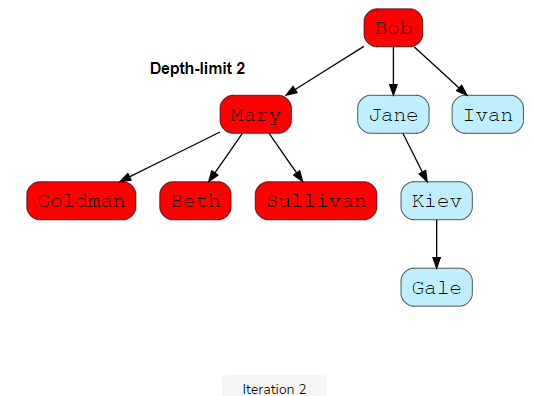
#### Figure Explanation

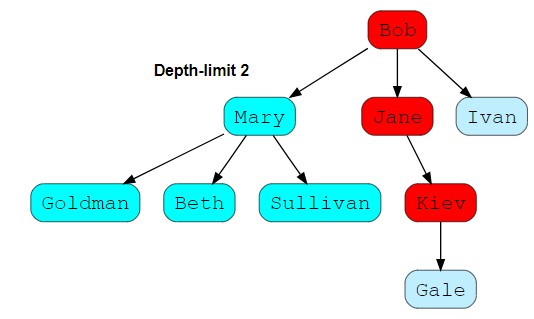
* We follow normal [DFS](https://www.educative.io/answers/what-is-depth-first-search) and start with the root node 1 at level 0.
* We mark it visited and add its children nodes, 2 and 3, to the stack. We increment our level by 1.
* Node 2 is at the top of our stack. We add its children, 4 and 5, to our stack. We increment our level by 1. Our level counter is now 2.
* Node 4 is at the top of our stack. We can add its child node 6 to our stack. However, doing so would exceed our level counter, so we ignore Node 6.
* Node 5 is at the top of our stack. It has no children to append to our stack.
* Node 5 is our desired result, so the algorithm stops

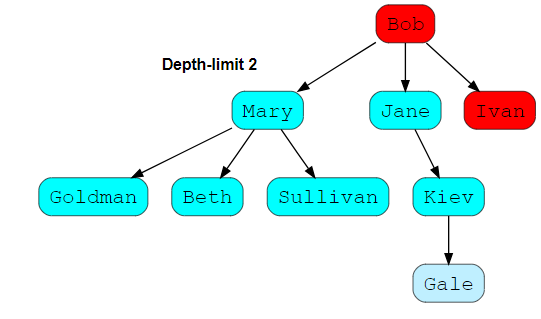
**ITERATIVE DEEPENING SEARCH**

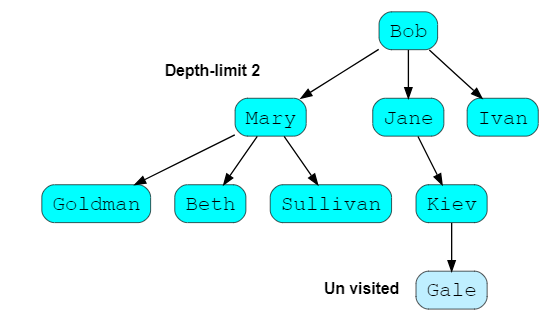
**Iterative Deepening Search (IDS)** is an iterative graph searching strategy that takes advantage of the completeness of the Breadth-First Search (BFS) strategy but uses much less memory in each iteration (similar to **Depth-First Search)**

IDS achieves the desired completeness by enforcing a depth-limit on DFS that mitigates the possibility of getting stuck in an infinite or a very long branch. It searches each branch of a node from left to right until it reaches the required depth. Once it has, IDS goes back to the root node and explores a different branch that is similar to **DFS.**



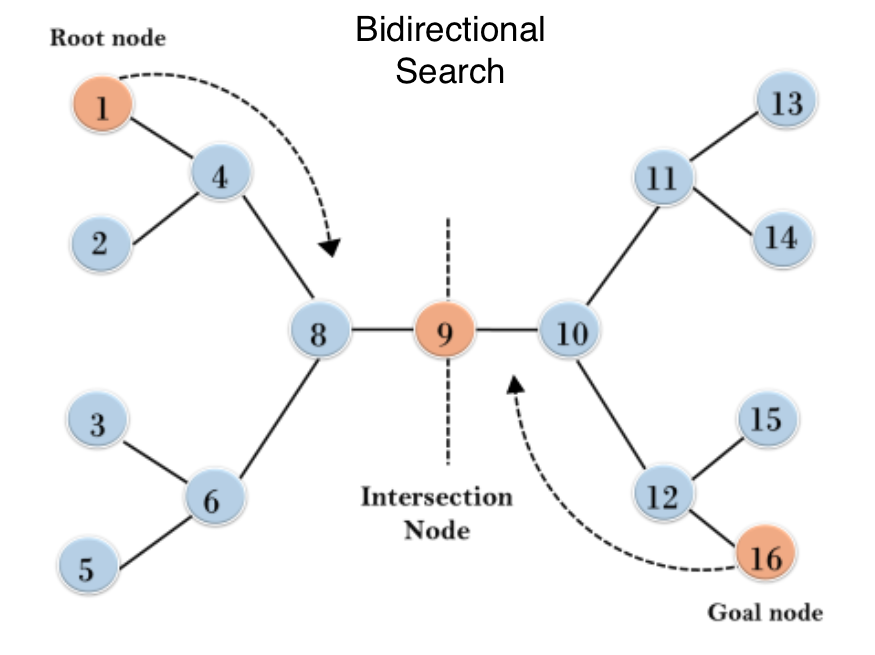




**BIDIRECTIONAL SEARCH**

**Bidirectional Search** is [Graph Search Algorithm](https://iq.opengenus.org/tag/graph-algorithm/) where two graph traversals (BFS) take place at the same time and is used to find the shortest distance between a fixed start vertex and end vertex. It is a faster approach, reduces the time required for traversing the graph. It can be used for other applications as well.

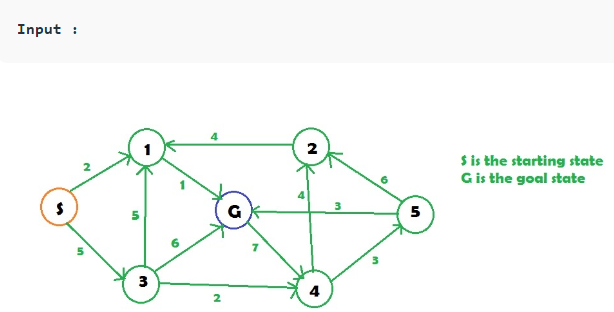
It significantly reduces the amount of exploration done. It is implemented using the Breadth First Search (BFS) Algorithm. (If you don't know what BFS is refer to [**this**](https://iq.opengenus.org/bfs-graph-traversal/) article first). BFS is run simultaneously on two vertices - the start and the end vertex. One single BFS tree is now replaced by two sub trees, and the search is terminated when the two trees intersect.

****

**UNIFORM COST SEARCH**

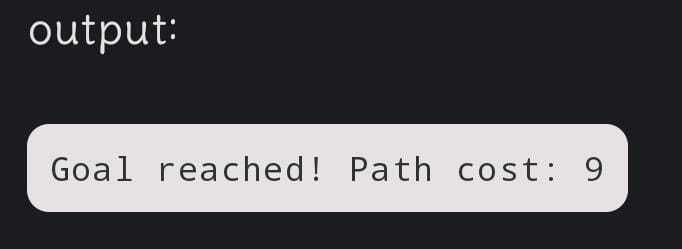
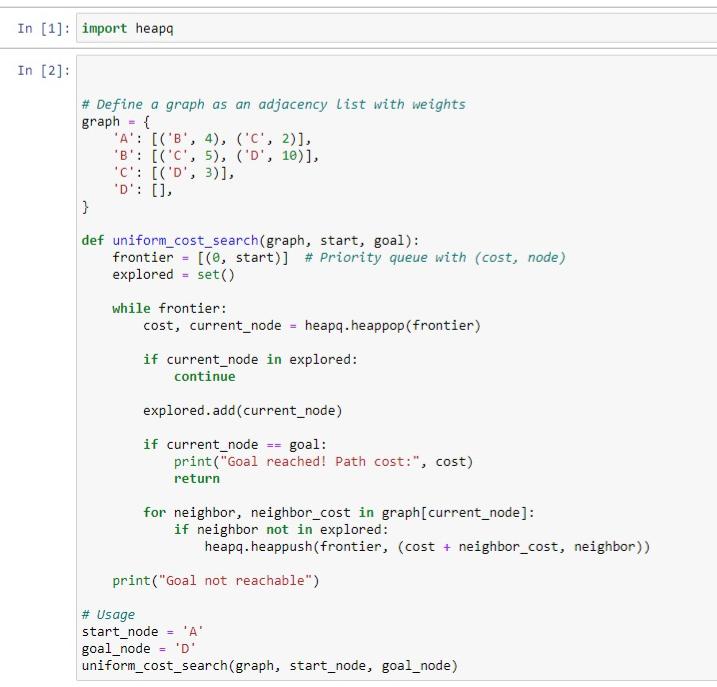
[**Uniform-Cost Search**](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Practical_optimizations_and_infinite_graphs) is a variant of **[Dijikstra’s](https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/)** [algorithm](https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/). Here, instead of inserting all vertices into a priority queue, we insert only the source, then one by one insert when needed. In every step, we check if the item is already in the priority queue (using the visited array). If yes, we perform the decrease key, else we insert it.

This variant of **Dijkstra** is useful for infinite graphs and that graph which are too large to represent in memory. Uniform-Cost Search is mainly used in **Artificial Intelligence.**

****

****

**UNIFORM COST SEARCH PROGRAM**

****