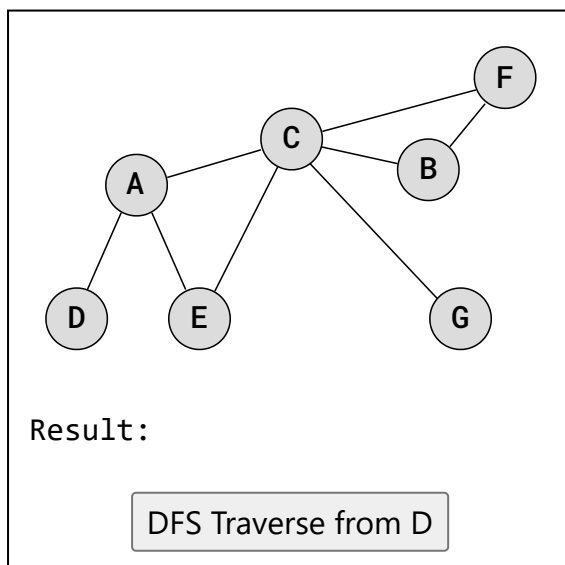


DSA Graphs Traversal

[< Previous](#)[Next >](#)

Graphs Traversal

To traverse a Graph means to start in one vertex, and go along the edges to visit other vertices until all vertices, or as many as possible, have been visited.



Understanding how a Graph can be traversed is important for understanding how algorithms that run on Graphs work.



DFS is usually implemented using a Stack or by the use of recursion (which utilizes the call stack), while BFS is usually implemented using a Queue.

The **Call Stack** keeps functions running in the correct order.

If for example FunctionA calls FunctionB, FunctionB is placed on top of the call stack and starts running. Once FunctionB is finished, it is removed from the stack, and then FunctionA resumes its work.

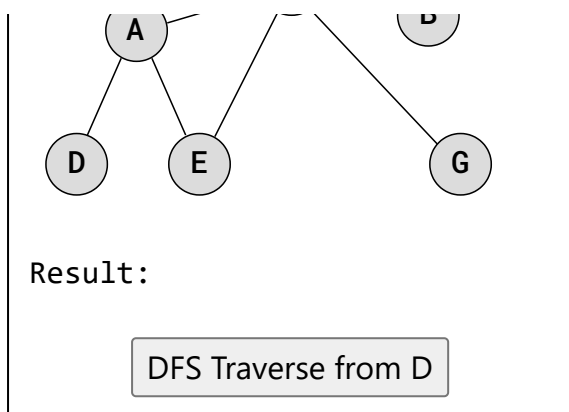
Depth First Search Traversal

Depth First Search is said to go "deep" because it visits a vertex, then an adjacent vertex, and then that vertex' adjacent vertex, and so on, and in this way the distance from the starting vertex increases for each recursive iteration.

How it works:

1. Start DFS traversal on a vertex.
2. Do a recursive DFS traversal on each of the adjacent vertices as long as they are not already visited.

Run the animation below to see how Depth First Search (DFS) traversal runs on a specific Graph, starting in vertex D (it is the same as the previous animation).



The DFS traversal starts in vertex D, marks vertex D as visited. Then, for every new vertex visited, the traversal method is called recursively on all adjacent vertices that have not been visited yet. So when vertex A is visited in the animation above, vertex C or vertex E (depending on the implementation) is the next vertex where the traversal continues.

Example

Python:

```

1  class Graph:
2      def __init__(self, size):
3          self.adj_matrix = [[0] * size for _ in range(size)]
4          self.size = size
5          self.vertex_data = [''] * size
6
7      def add_edge(self, u, v):
8          if 0 <= u < self.size and 0 <= v < self.size:
9              self.adj_matrix[u][v] = 1
10             self.adj_matrix[v][u] = 1
11
12     def add_vertex_data(self, vertex, data):
13         if 0 <= vertex < self.size:
14             self.vertex_data[vertex] = data
15
16     def print_graph(self):
17         print("Adjacency Matrix:")
18         for row in self.adj_matrix:
19             print(' '.join(map(str, row)))
20         print("\nVertex Data:")

```



Tutorials ▼

References ▼

Exercises ▼



Sign In

SS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C C

```
36
37  g = Graph(7)
38
39  g.add_vertex_data(0, 'A')
40  g.add_vertex_data(1, 'B')
41  g.add_vertex_data(2, 'C')
42  g.add_vertex_data(3, 'D')
43  g.add_vertex_data(4, 'E')
44  g.add_vertex_data(5, 'F')
45  g.add_vertex_data(6, 'G')
46
47  g.add_edge(3, 0)  # D - A
48  g.add_edge(0, 2)  # A - C
49  g.add_edge(0, 3)  # A - D
50  g.add_edge(0, 4)  # A - E
51  g.add_edge(4, 2)  # E - C
52  g.add_edge(2, 5)  # C - F
53  g.add_edge(2, 1)  # C - B
54  g.add_edge(2, 6)  # C - G
55  g.add_edge(1, 5)  # B - F
56
57  g.print_graph()
58
59  print("\nDepth First Search starting from vertex D:")
```



Line 60: The DFS traversal starts when the `dfs()` method is called.

Line 33: The `visited` array is first set to `false` for all vertices, because no vertices are visited yet at this point.

Line 35: The `visited` array is sent as an argument to the `dfs_util()` method. When the `visited` array is sent as an argument like this, it is actually just a reference to the `visited` array that is sent to the `dfs_util()` method, and not the actual array with the values inside. So there is always just one `visited` array in our program, and the `dfs_util()` method can make changes to it as nodes are visited (line 25).

Line 28-30: For the current vertex `v`, all adjacent nodes are called recursively if they are not already visited.

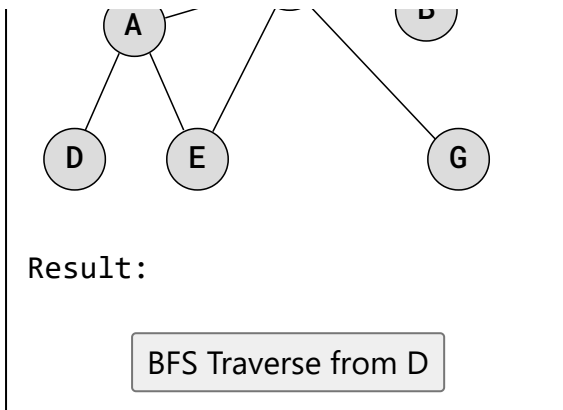
Breadth First Search Traversal

Breadth First Search visits all adjacent vertices of a vertex before visiting neighboring vertices to the adjacent vertices. This means that vertices with the same distance from the starting vertex are visited before vertices further away from the starting vertex are visited.

How it works:

1. Put the starting vertex into the queue.
2. For each vertex taken from the queue, visit the vertex, then put all unvisited adjacent vertices into the queue.
3. Continue as long as there are vertices in the queue.

Run the animation below to see how Breadth First Search (BFS) traversal runs on a specific Graph, starting in vertex D.



As you can see in the animation above, BFS traversal visits vertices the same distance from the starting vertex, before visiting vertices further away. So for example, after visiting vertex A, vertex E and C are visited before visiting B, F and G because those vertices are further away.

Breadth First Search traversal works this way by putting all adjacent vertices in a queue (if they are not already visited), and then using the queue to visit the next vertex.

This code example for Breadth First Search traversal is the same as for the Depth First Search code example above, except for the `bfs()` method:

Example

Python:

```
def bfs(self, start_vertex_data):
    queue = [self.vertex_data.index(start_vertex_data)]
    visited = [False] * self.size
    visited[queue[0]] = True

    while queue:
        current_vertex = queue.pop(0)
        print(self.vertex_data[current_vertex], end=' ')

        for i in range(self.size):
            if self.adj_matrix[current_vertex][i] == 1 and not visi
```

[Try it Yourself »](#)

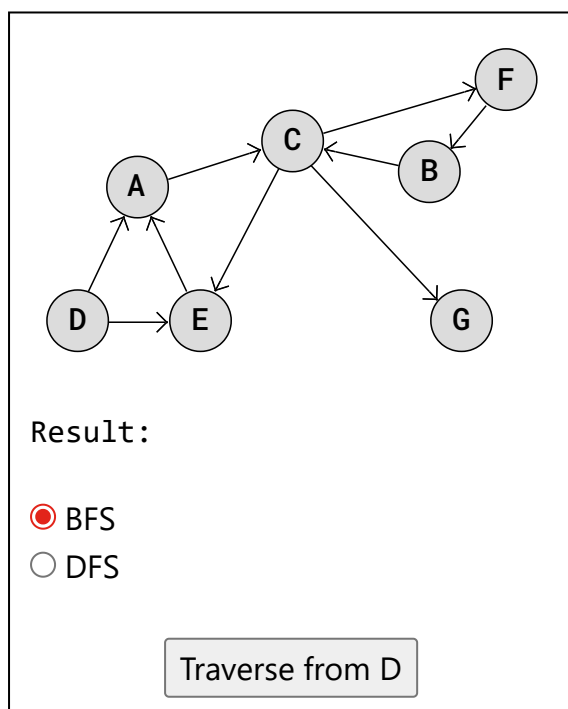
Line 2-4: The `bfs()` method starts by creating a queue with the start vertex inside, creating a `visited` array, and setting the start vertex as visited.

Line 6-13: The BFS traversal works by taking a vertex from the queue, printing it, and adding adjacent vertices to the queue if they are not visited yet, and then continue to take vertices from the queue in this way. The traversal finishes when the last element in the queue has no unvisited adjacent vertices.

DFS and BFS Traversal of a Directed Graph

Depth first and breadth first traversals can actually be implemented to work on directed Graphs (instead of undirected) with just very few changes.

Run the animation below to see how a directed Graph can be traversed using DFS or BFS.



To go from traversing a directed Graph instead of an undirected Graph, we just need to remove the last line in the `add_edge()` method:



```
self.adj_matrix[u][v] = 1
self.adj_matrix[v][u] = 1
```

We must also take care when we build our Graph because the edges are now directed.

The code example below contains both BFS and DFS traversal of the directed Graph from the animation above:

Example

Python:

```
class Graph:
    def __init__(self, size):
        self.adj_matrix = [[0] * size for _ in range(size)]
        self.size = size
        self.vertex_data = [''] * size

    def add_edge(self, u, v):
        if 0 <= u < self.size and 0 <= v < self.size:
            self.adj_matrix[u][v] = 1
            #self.adj_matrix[v][u] = 1

    def add_vertex_data(self, vertex, data):
        if 0 <= vertex < self.size:
            self.vertex_data[vertex] = data

    def print_graph(self):
        print("Adjacency Matrix:")
        for row in self.adj_matrix:
            print(' '.join(map(str, row)))
        print("\nVertex Data:")
        for vertex, data in enumerate(self.vertex_data):
            print(f"Vertex {vertex}: {data}")

    def dfs_util(self, v, visited):
        visited[v] = True
```



```

        self.dfs_util(i, visited)

    def dfs(self, start_vertex_data):
        visited = [False] * self.size

        start_vertex = self.vertex_data.index(start_vertex_data)
        self.dfs_util(start_vertex, visited)

    def bfs(self, start_vertex_data):
        queue = [self.vertex_data.index(start_vertex_data)]
        visited = [False] * self.size
        visited[queue[0]] = True

        while queue:
            current_vertex = queue.pop(0)
            print(self.vertex_data[current_vertex], end=' ')

            for i in range(self.size):
                if self.adj_matrix[current_vertex][i] == 1 and not visited[i]:
                    queue.append(i)
                    visited[i] = True

g = Graph(7)

g.add_vertex_data(0, 'A')
g.add_vertex_data(1, 'B')
g.add_vertex_data(2, 'C')
g.add_vertex_data(3, 'D')
g.add_vertex_data(4, 'E')
g.add_vertex_data(5, 'F')
g.add_vertex_data(6, 'G')

g.add_edge(3, 0) # D -> A
g.add_edge(3, 4) # D -> E
g.add_edge(4, 0) # E -> A
g.add_edge(0, 2) # A -> C
g.add_edge(2, 5) # C -> F
g.add_edge(2, 6) # C -> G
g.add_edge(5, 1) # F -> B

```

[Tutorials ▼](#)[References ▼](#)[Exercises ▼](#)[Sign In](#)[SS](#) [JAVASCRIPT](#) [SQL](#) [PYTHON](#) [JAVA](#) [PHP](#) [HOW TO](#) [W3.CSS](#) [C](#) [C](#)

```
print("\nDepth First Search starting from vertex D:")
g.dfs('D')

print("\n\nBreadth First Search starting from vertex D:")
g.bfs('D')
```

[Try it Yourself »](#)

Now that we have looked at two basic algorithms for how to traverse Graphs, we will use the next pages to see how other algorithms can run on the Graph data structure.

[◀ Previous](#)[Sign in to track progress](#)[Next >](#)

Get Certified!

Document your skills with all of
W3Schools Certificates

~~\$1,995~~

\$499

Save 75% 🎁



COLOR PICKER



[Tutorials ▼](#)[References ▼](#)[Exercises ▼](#)[Sign In](#)[SS](#)[JAVASCRIPT](#)[SQL](#)[PYTHON](#)[JAVA](#)[PHP](#)[HOW TO](#)[W3.CSS](#)[C](#)[C](#)[PLUS](#)[SPACES](#)[GET CERTIFIED](#)[FOR TEACHERS](#)[FOR BUSINESS](#)[CONTACT US](#)

Top Tutorials

[HTML Tutorial](#)
[CSS Tutorial](#)
[JavaScript Tutorial](#)
[How To Tutorial](#)
[SQL Tutorial](#)
[Python Tutorial](#)
[W3.CSS Tutorial](#)
[Bootstrap Tutorial](#)
[PHP Tutorial](#)
[Java Tutorial](#)
[C++ Tutorial](#)
[jQuery Tutorial](#)

Top References

[HTML Reference](#)
[CSS Reference](#)
[JavaScript Reference](#)
[SQL Reference](#)
[Python Reference](#)
[W3.CSS Reference](#)
[Bootstrap Reference](#)
[PHP Reference](#)
[HTML Colors](#)
[Java Reference](#)
[AngularJS Reference](#)
[jQuery Reference](#)

Top Examples

[HTML Examples](#)
[CSS Examples](#)
[JavaScript Examples](#)
[How To Examples](#)

Get Certified

[HTML Certificate](#)
[CSS Certificate](#)
[JavaScript Certificate](#)
[Front End Certificate](#)

[Tutorials ▼](#)[References ▼](#)[Exercises ▼](#)[Sign In](#)[SS](#)[JAVASCRIPT](#)[SQL](#)[PYTHON](#)[JAVA](#)[PHP](#)[HOW TO](#)[W3.CSS](#)[C](#)[C](#)[XML Examples](#)
[jQuery Examples](#)[C# Certificate](#)
[XML Certificate](#)[FORUM](#) [ABOUT](#) [ACADEMY](#)

W3Schools is optimized for learning and training. Examples might be simplified to improve reading and learning.

Tutorials, references, and examples are constantly reviewed to avoid errors, but we cannot warrant full correctness

of all content. While using W3Schools, you agree to have read and accepted our [terms of use](#), [cookies](#) and [privacy policy](#).

[Copyright 1999-2026](#) by Refsnes Data. All Rights Reserved. W3Schools is Powered by [W3.CSS](#).