Malla Reddy University

I B.Tech II Semester – CSE/AIML/CS/DS/IT/IOT

Question Bank

Data Structures and Its Applications

Course Code: MR22-1CS0105

DSA MID–2 QUESTIONS AND ANSWERS

UNIT - V

DONE AND PRESENTED BY-

TEAM VISION

Contact:-
teamvision1729@gmail.com

UNIT-V

1. Define a graph and its characteristics. How is a graph different from other data structures? Explain with suitable examples.
2. Describe the characteristics of a graph. Discuss the concepts of vertices, edges, directed and undirected graphs, weighted and unweighted graphs.
3. Write the difference between linear and non-linear data structures. Explain with suitable examples discussing pros and cons.
4. Explain Depth First Search with an example.
5. Explain Breadth First Search with an example.
6. Compare and contrast BFS and DFS algorithms in terms of their implementation, traversal order, and the data structures used. Discuss the advantages and disadvantages of each algorithm.

UNIT-5

1. **Define a graph and its characteristics. How is a graph different from other data structures? Explain with suitable examples.**

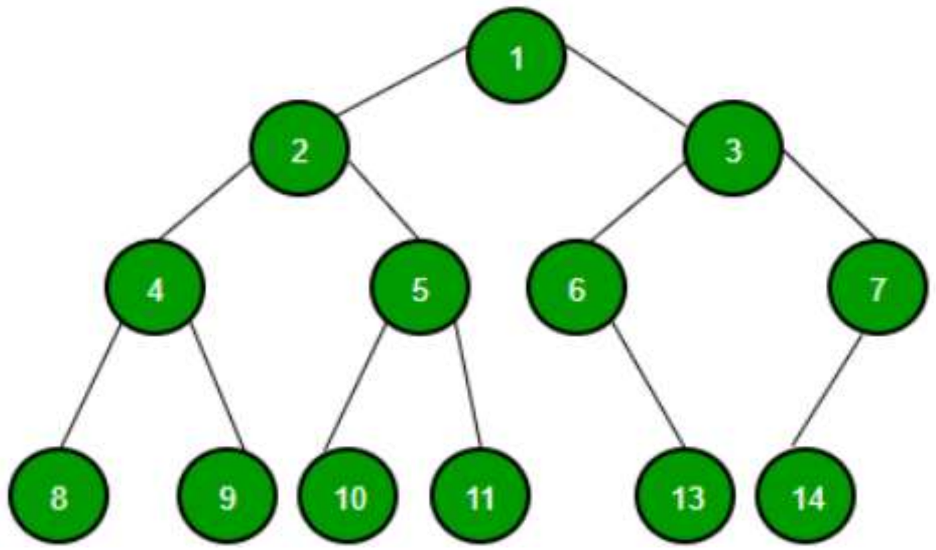
Ans:

A graph is a non-linear data structure consisting of a collection of nodes (vertices) connected by edges. It is used to represent relationships between objects or entities. In a graph, nodes can be connected to any number of other nodes through edges, allowing for complex connectivity patterns. Graphs may have cycles, but it doesn’t follow any hierarchy.

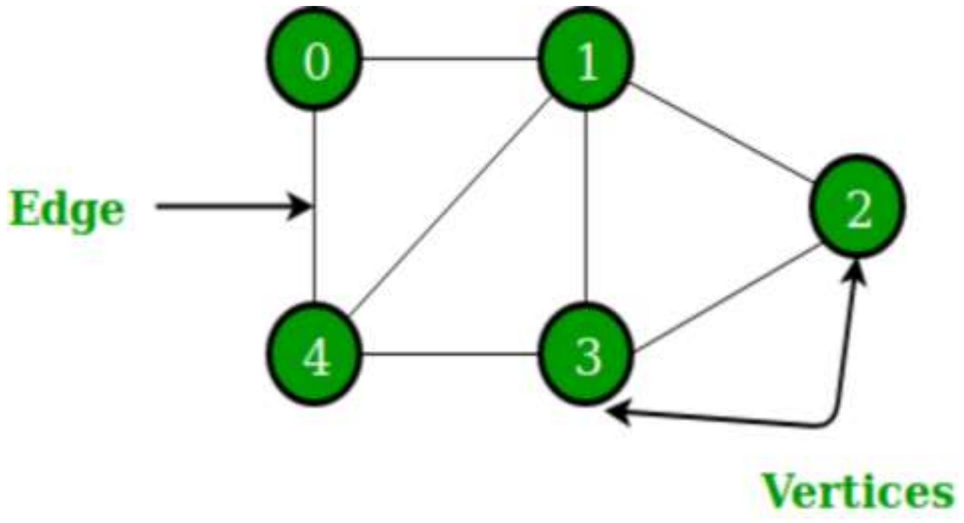
Characteristics of a Graph:

- **Vertices (Nodes):** Graphs are composed of vertices or nodes that represent objects or entities. Each node can hold additional information beyond its identification.
- **Edges:** Edges are connections or relationships between nodes in a graph. They represent the associations or links between entities and can be directed or undirected. Directed edges have a specific direction, while undirected edges do not.
- **Connectivity:** Graphs capture the connectivity between nodes, showing how different nodes are related or connected. The arrangement of edges determines the structure of the graph and the relationship between its components.
- **Weighted Edges:** Edges in a graph can have associated weights or values. These weights may represent factors such as distances, costs, or strengths of relationships between nodes.
- **Cyclic or Acyclic:** A graph can be cyclic or acyclic. A cyclic graph contains at least one cycle, which is a closed path that starts and ends at the same node. An acyclic graph does not contain any cycles.
- **Representation:** Graphs can be represented in various ways, such as adjacency matrix, adjacency list, or edge list. The choice of representation depends on the specific use case and desired operations on the graph.

#Example (Comparison)



This is an Example figure for Tree Data Structure



This is an Example figure for Graph Data Structure

2. Describe the characteristics of a graph. Discuss the concepts of vertices, edges, directed and undirected graphs, weighted and unweighted graphs.

Ans:

Characteristics of a Graph:

- **Vertices (Nodes):** Graphs are composed of vertices or nodes that represent objects or entities. Each node can hold additional information beyond its identification.
- **Edges:** Edges are connections or relationships between nodes in a graph. They represent the associations or links between entities and can be directed or undirected. Directed edges have a specific direction, while undirected edges do not.
- **Connectivity:** Graphs capture the connectivity between nodes, showing how different nodes are related or connected. The arrangement of edges determines the structure of the graph and the relationship between its components.
- **Weighted Edges:** Edges in a graph can have associated weights or values. These weights may represent factors such as distances, costs, or strengths of relationships between nodes.
- **Cyclic or Acyclic:** A graph can be cyclic or acyclic. A cyclic graph contains at least one cycle, which is a closed path that starts and ends at the same node. An acyclic graph does not contain any cycles.
- **Representation:** Graphs can be represented in various ways, such as adjacency matrix, adjacency list, or edge list. The choice of representation depends on the specific use case and desired operations on the graph.

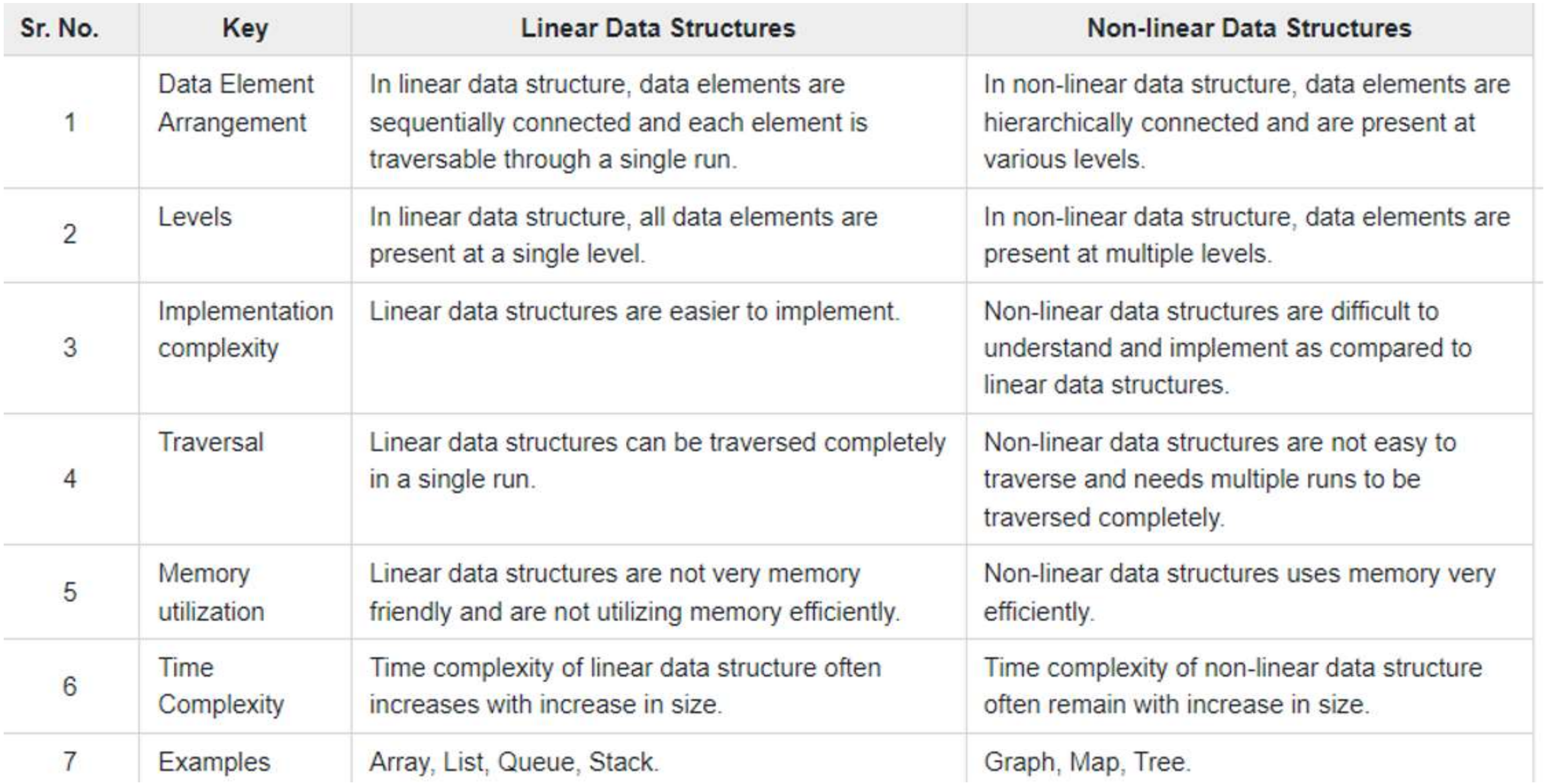
The concepts of –

- **Vertices:** The point where the edges meet in a graph is called Vertices or Node. A vertex can be a object or element.
- **Edges:** The connection between the vertices is known as edges. The edges are two types they are
    1) Unidirectional / Bidirectional
    2) Directed / Unidirectional

- **Directed Graphs:** also called Digraph or Unidirectional graphs. Each edge in a directed graph is connected from one vertex to another vertex which a specific direction (→)
- **Undirected Graphs:** It is a graph where each edge doesn't have a specific direction. It is also called a bidirectional graph in which we can traverse in both directions.
- **Weighted Graph:** In a weighted graph, each edge is assigned a numerical value or weight. The weight represents additional information associated with the relationship between vertices, such as distance, cost, or strength.
- **Unweighted Graph:** In an unweighted graph, all edges have equal weight or no weight associated with them. The presence or absence of an edge indicates the existence or absence of a relationship between vertices, without any numerical value assigned.

3. Write the difference between linear and non-linear data structures. Explain with suitable examples discussing pros and cons.

Ans:



Sr. No.	Key	Linear Data Structures	Non-linear Data Structures
1	Data Element Arrangement	In linear data structure, data elements are sequentially connected and each element is traversable through a single run.	In non-linear data structure, data elements are hierarchically connected and are present at various levels.
2	Levels	In linear data structure, all data elements are present at a single level.	In non-linear data structure, data elements are present at multiple levels.
3	Implementation complexity	Linear data structures are easier to implement.	Non-linear data structures are difficult to understand and implement as compared to linear data structures.
4	Traversal	Linear data structures can be traversed completely in a single run.	Non-linear data structures are not easy to traverse and needs multiple runs to be traversed completely.
5	Memory utilization	Linear data structures are not very memory friendly and are not utilizing memory efficiently.	Non-linear data structures uses memory very efficiently.
6	Time Complexity	Time complexity of linear data structure often increases with increase in size.	Time complexity of non-linear data structure often remain with increase in size.
7	Examples	Array, List, Queue, Stack.	Graph, Map, Tree.

Suitable Examples:

- **Linear Data Structure Example: Array**
  - Pros:

    - Constant time access to elements using indexes.

    - Efficient for iterating through elements sequentially.

    - Simple and straightforward implementation.
  - Cons:

    - Fixed-size, making it challenging to resize dynamically.

    - Insertion and deletion operations may require shifting elements, leading to time-consuming operations.
- **Non-linear Data Structure Example: Binary Search Tree (BST)**
  - Pros:

    - Efficient for searching, insertion, and deletion operations in a sorted collection.

    - Provides quick access to the minimum and maximum elements.

    - Can be used for ordered traversal of elements.
  - Cons:

    - Degenerate cases may lead to skewed trees, resulting in inefficient operations.

    - Balancing the tree requires additional operations and algorithms.

    - Not suitable for unordered collections or when maintaining insertion order is important.

4. Explain Depth First Search with an example.

Ans.

- Depth-first search (DFS), is an algorithm for traversal on graph or tree data structures. It can be implemented easily using recursion and data structures like dictionaries and sets.
- The only purpose of this algorithm is to visit all the vertex of the graph avoiding cycles. To avoid processing a node more than once, use a Boolean visited array. A graph can have more than one DFS traversal.

A standard DFS implementation puts each vertex of the graph into one of two categories:

- Visited
- Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

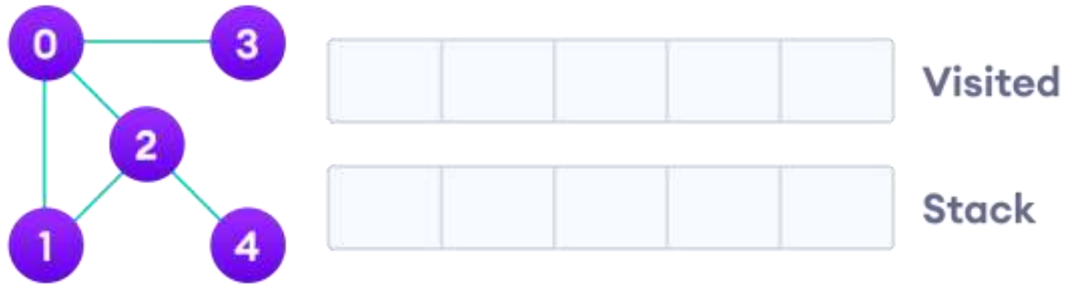
Depth First Search Algorithm:-

The DFS algorithm works as follows:

1. Start by putting any one of the graph's vertices on top of a stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
4. Keep repeating steps 2 and 3 until the stack is empty.

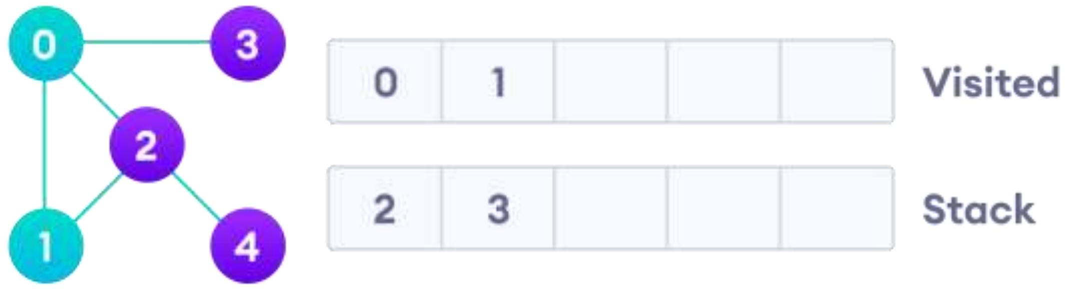
Let's see how the Depth First Search algorithm works with an example. We use an undirected graph with 5 vertices.

Initially stack and visited arrays are empty.



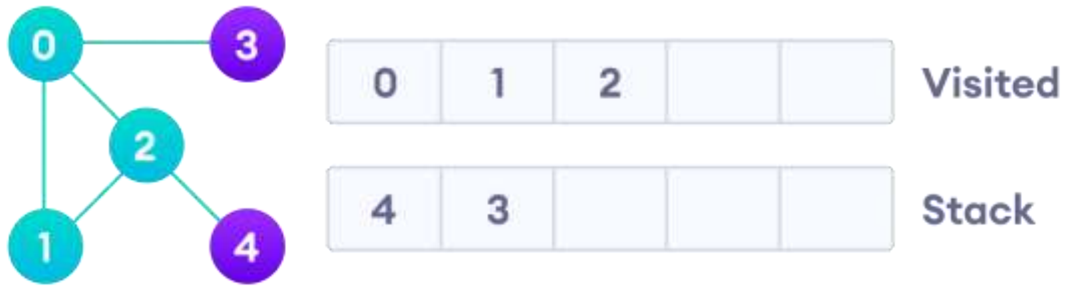
We start from vertex 0, the DFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.

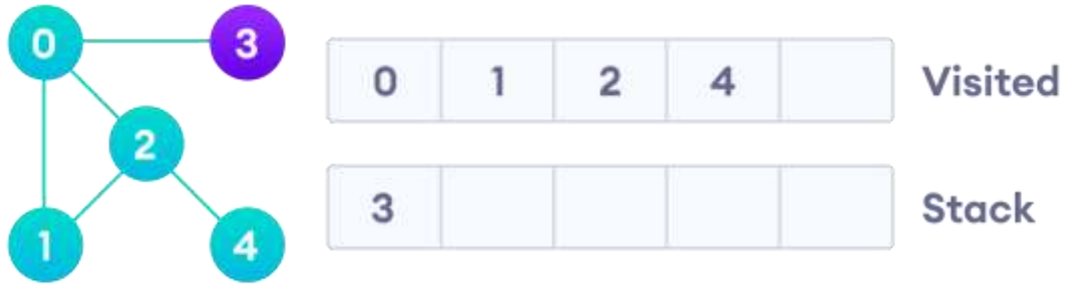
Next, we visit the element at the top of stack i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.



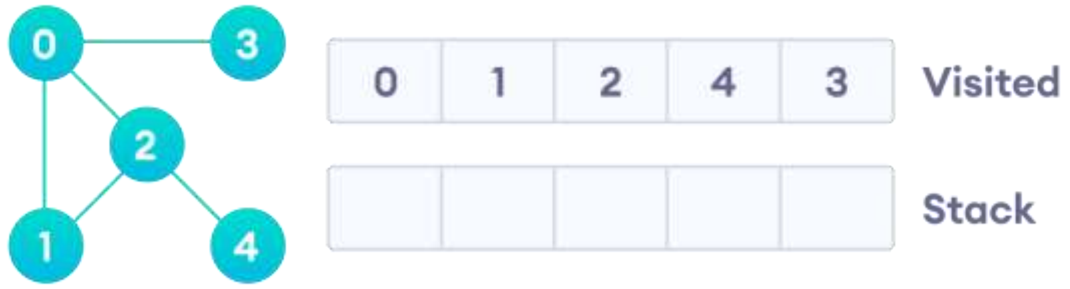
Visit the element at the top of stack.

Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.



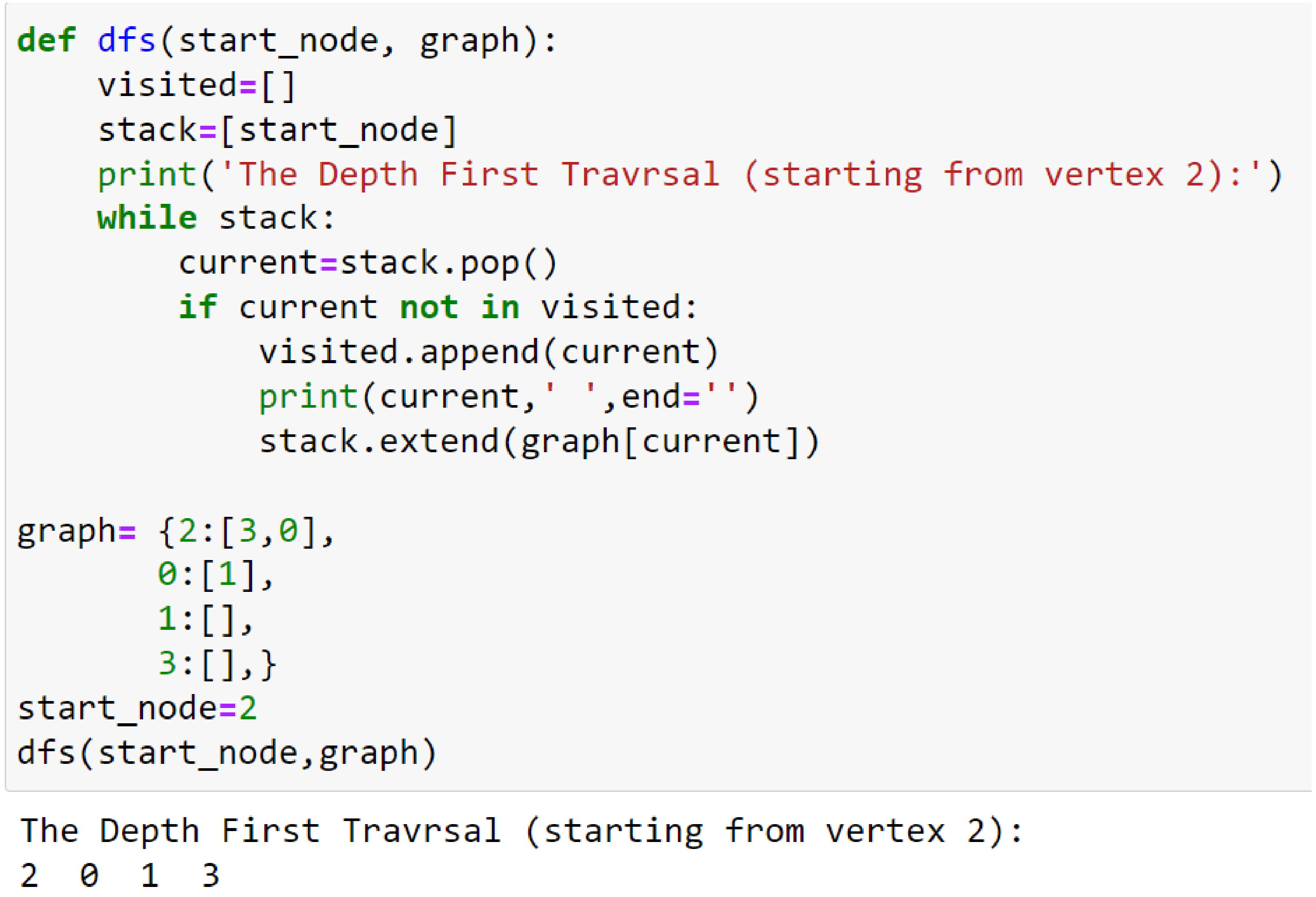


After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First Traversal of the graph.



Now, Stack becomes empty, which means we have visited all the nodes and our DFS traversal ends.

Program for DFS:-



```python
def dfs(start_node, graph):
    visited=[]
    stack=[start_node]
    print('The Depth First Travrsal (starting from vertex 2):')
    while stack:
        current=stack.pop()
        if current not in visited:
            visited.append(current)
            print(current,'  ',end='')
            stack.extend(graph[current])

graph= {2:[3,0],
       0:[1],
       1:[],
       3:[],}
start_node=2
dfs(start_node,graph)
```

```
The Depth First Travrsal (starting from vertex 2):
2  0  1  3
```

5. Explain Breadth-First Search with an example.

Ans.

Breadth-first search:

- BFS stands for ***Breadth First Search***. It is also known as **level order traversal**. The Queue data structure is used for the Breadth First Search traversal. When we use the BFS algorithm for the traversal in a graph, we can consider any node as a root node.

A standard BFS implementation puts each vertex of the graph into one of two categories:

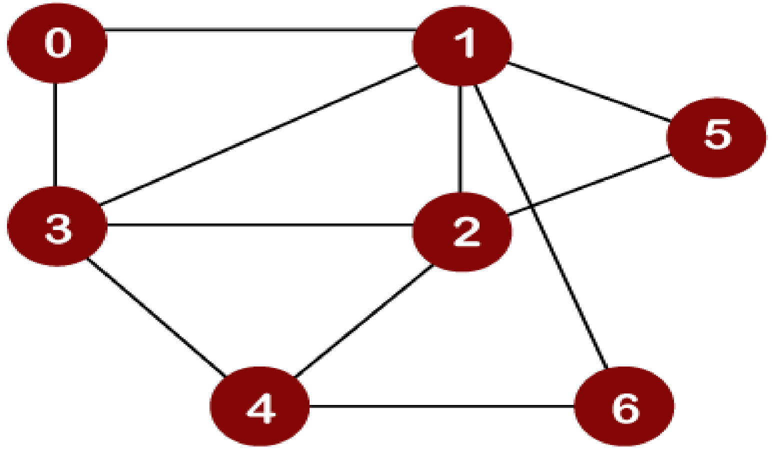
- Visited
- Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

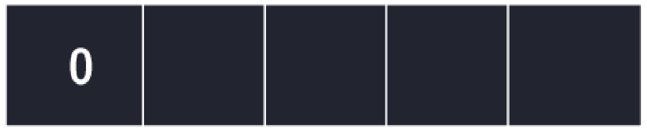
BFS Algorithm:-

- Start by putting any one of the graph's vertices at the back of a queue.
- Take the front item of the queue and add it to the visited list.
- Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
- Keep repeating steps 2 and 3 until the queue is empty.
- The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node

- **Let's consider the below graph for the breadth first search traversal.**

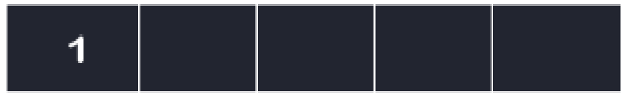


Suppose we consider node 0 as a root node. Therefore, the traversing would be started from node 0.



Once node 0 is removed from the Queue, it gets printed and marked as a ***visited node.***

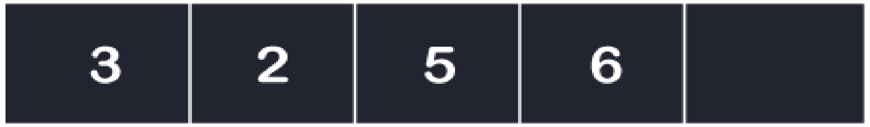
Once node 0 gets removed from the Queue, then the adjacent nodes of node 0 would be inserted in a Queue as shown below:



Result : 0

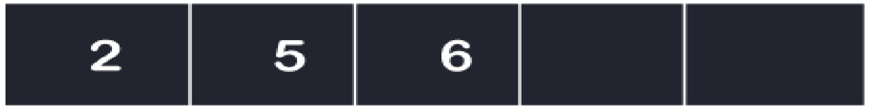
Now the node 1 will be removed from the Queue; it gets printed and marked as a visited node

Once node 1 gets removed from the Queue, then all the adjacent nodes of a node 1 will be added in a Queue. The adjacent nodes of node 1 are 0, 3, 2, 6, and 5. But we have to insert only unvisited nodes in a Queue. Since nodes 3, 2, 6, and 5 are unvisited; therefore, these nodes will be added in a Queue as shown below:



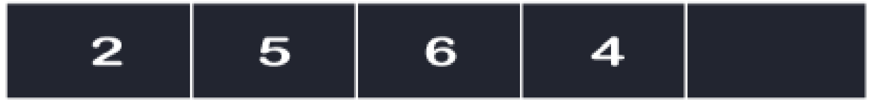
Result : 0 , 1

The next node is 3 in a Queue. So, node 3 will be removed from the Queue, it gets printed and marked as visited as shown below:



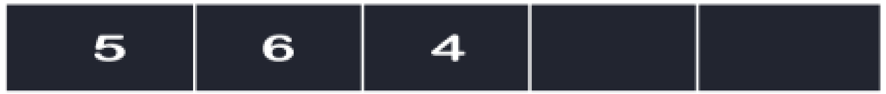
Result : 0, 1, 3

Once node 3 gets removed from the Queue, then all the adjacent nodes of node 3 except the visited nodes will be added in a Queue. The adjacent nodes of node 3 are 0, 1, 2, and 4. Since nodes 0, 1 are already visited, and node 2 is present in a Queue; therefore, we need to insert only node 4 in a Queue.



Result : 0, 1, 3

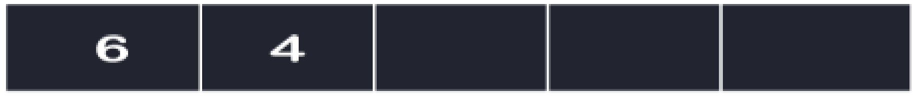
Now, the next node in the Queue is 2. So, 2 would be deleted from the Queue. It gets printed and marked as visited as shown below:



Result : 0, 1, 3, 2,

Once node 2 gets removed from the Queue, then all the adjacent nodes of node 2 except the visited nodes will be added in a Queue. The adjacent nodes of node 2 are 1, 3, 5, 6, and 4. Since nodes 1 and 3 have already been visited, and 4, 5, 6 are already added in the Queue; therefore, we do not need to insert any node in the Queue.

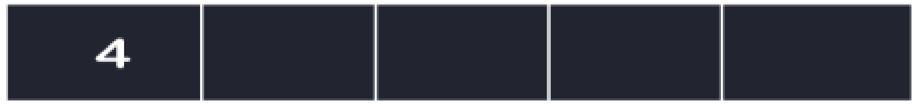
The next element is 5. So, 5 would be deleted from the Queue. It gets printed and marked as visited as shown below:



Result : 0, 1, 3, 2, 5

Once node 5 gets removed from the Queue, then all the adjacent nodes of node 5 except the visited nodes will be added in the Queue. The adjacent nodes of node 5 are 1 and 2. Since both the nodes have already been visited; therefore, there is no vertex to be inserted in a Queue.

The next node is 6. So, 6 would be deleted from the Queue. It gets printed and marked as visited as shown below:



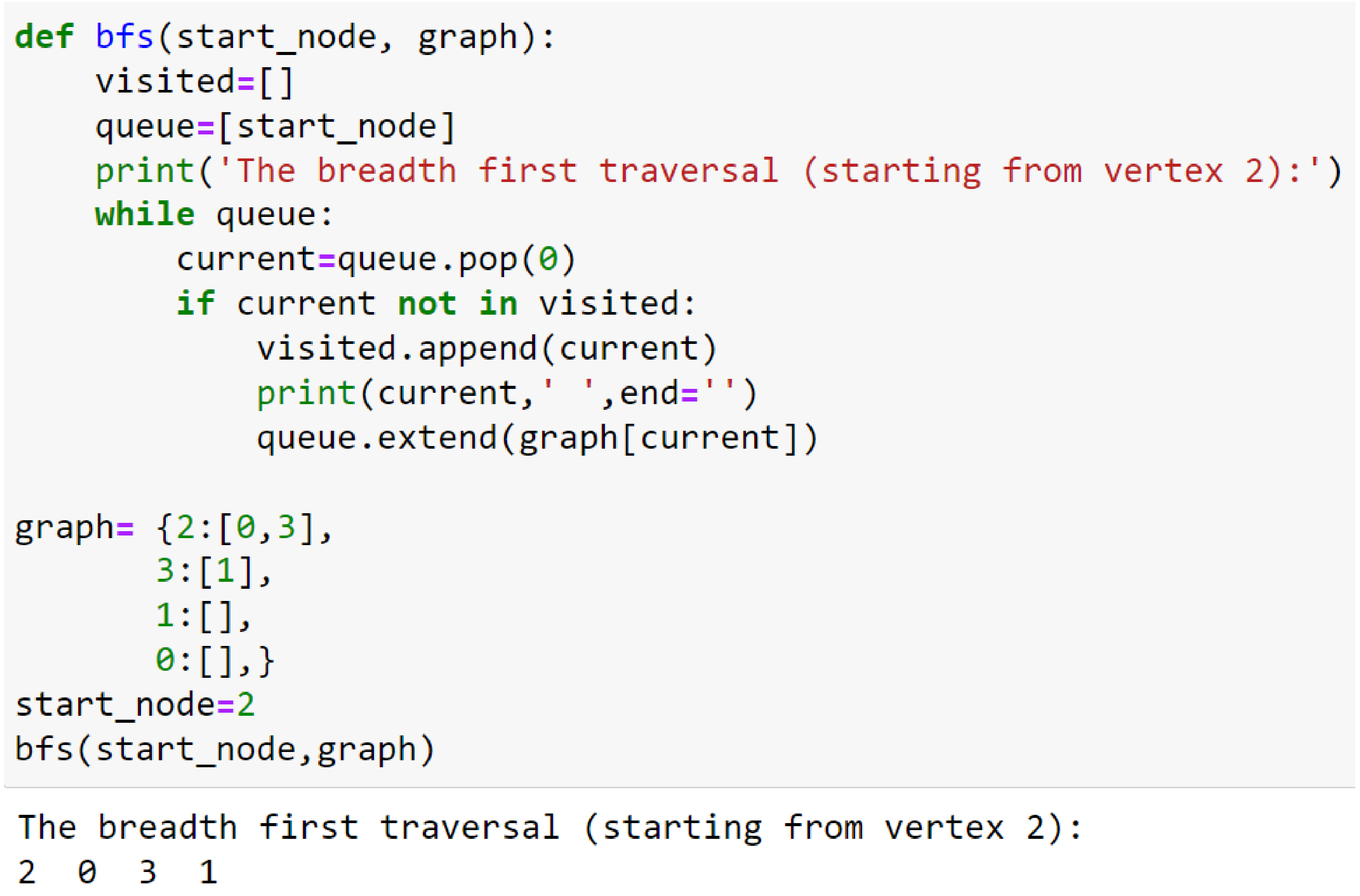
Result : 0, 1, 3, 2, 5, 6

Once the node 6 gets removed from the Queue, then all the adjacent nodes of node 6 except the visited nodes will be added in the Queue. The adjacent nodes of node 6 are 1 and 4. Since the node 1 has already been visited and node 4 is already added in the Queue; therefore, there is not vertex to be inserted in the Queue.

The next element in the Queue is 4. So, 4 would be deleted from the Queue. It gets printed and marked as visited.

Since the queue is empty, we have completed the Breadth First Traversal of the graph.

Program for BFS :-



```python
def bfs(start_node, graph):
    visited=[]
    queue=[start_node]
    print('The breadth first traversal (starting from vertex 2):')
    while queue:
        current=queue.pop(0)
        if current not in visited:
            visited.append(current)
            print(current,' ',end='')
            queue.extend(graph[current])

graph= {2:[0,3],
       3:[1],
       1:[],
       0:[],}
start_node=2
bfs(start_node,graph)
```

```
The breadth first traversal (starting from vertex 2):
2  0  3  1
```

6. Compare and contrast BFS and DFS algorithms in terms of their implementation, traversal order, and the data structures used. Discuss the advantages and disadvantages of each algorithm.

Ans.

S.No.	**Parameters**	**BFS**	**DFS**
1.	**Implementation**	Implemented using a queue data structure. The algorithm explores all the vertices at the same level before moving on to the next level.	Implemented using a stack data structure (or recursion). The algorithm explores as deep as possible before backtracking.
2.	**Traversal Order**	Visits vertices in increasing order of their distance from the starting vertex. It explores all the neighbors before moving to the next level.	Visits vertices in the order they are encountered. It explores as far as possible along each branch before backtracking.
3.	**Data Structures Used**	Uses a queue to store the nodes to visit. The queue operates on a "first-in, first-out“ (FIFO) principle.	Uses a stack (or recursion) to store the nodes to visit. The stack operates on a "lastin, first-out" (LIFO) principle.

Advantages and Disadvantages:

BFS:

Advantages:

- Guarantees the shortest path in an unweighted graph.
- Suitable for finding the shortest path or minimum steps to reach a target.

Disadvantages:

- Requires more memory to store the visited nodes and the queue.
- In graphs with a high branching factor, the queue can become large.

DFS:

Advantages:

- Requires less memory compared to BFS as it explores the graph deeply.
- Can be more efficient in graphs with a high branching factor or in cases where the search space is large but the solution is likely to be found deep in the graph.

Disadvantages:

- Doesn't guarantee the shortest path.
- Can get stuck in an infinite loop if the graph has cycles.