THE GRAPH

TRAVERSAL

THE GRAPH TRAVERSAL

THIS IS VERY SIMILAR TO
TREE TRAVERSAL

THE SAME 2 TYPES:

DEPTH-FIRST

BREADTH-FIRST

THE GRAPH TRAVERSAL

DEPTH-FIRST
BREADTH-FIRST

ONE ADDITIONAL WRINKLE

IN A TREE THERE IS ONLY ONE PATH FROM THE ROOT TO A SPECIFIC NODE

IN A GRAPH MULTIPLE PATHS CAN LEAD FROM ONE NODE TO ANOTHER

A GRAPH CAN ALSO HAVE CYCLES, SO THE SAME NODE CAN BE VISITED MULTIPLE TIMES

THE GRAPH TRAVERSAL

DEPTH-FIRST
BREADTH-FIRST

IN A GRAPH MULTIPLE PATHS CAN LEAD FROM ONE NODE TO ANOTHER

A GRAPH CAN ALSO HAVE CYCLES, SO THE SAME NODE CAN BE VISITED MULTIPLE TIMES

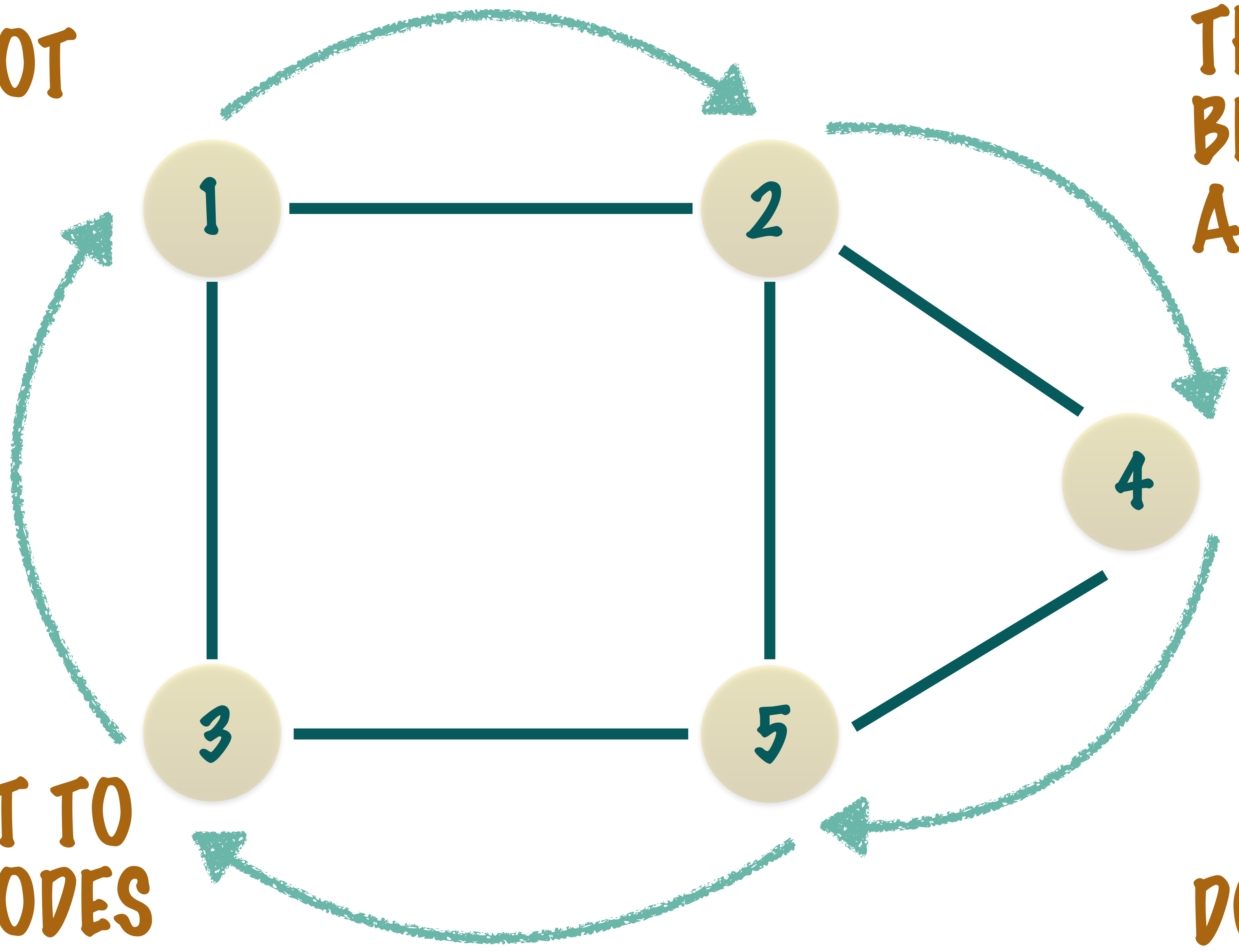
IN ORDER TO AVOID INFINITE LOOPING IN A GRAPH WE NEED TO KEEP TRACK OF THE NODES PREVIOUSLY VISITED

THE GRAPH

DEPTH-FIRST TRAVERSAL

NODE 1 SHOULD NOT BE VISITED AGAIN

THE VISITED LIST CAN BE A SIMPLE BOOLEAN ARRAY



VISITED = TRUE MEANS THE NODE HAS BEEN SEEN BEFORE

USE A VISITED LIST TO KEEP TRACK OF NODES ALREADY VISITED

DO NOT PROCESS THOSE NODES AGAIN

THE GRAPH

DEPTH-FIRST TRAVERSAL

LET'S SEE SOME CODE...

GRAPH - DEPTH FIRST TRAVERSAL

```java
public static void depthFirstTraversal(Graph graph, int[] visited, int currentVertex) {
    if (visited[currentVertex] == 1) {
        return;
    }
    visited[currentVertex] = 1;

    List<Integer> list = graph.getAdjacentVertices(currentVertex);
    for (int vertex : list) {
        depthFirstTraversal(graph, visited, vertex);
    }

    System.out.print(currentVertex + "->");
}
```

SPECIFIC THE GRAPH, VISITED NODE LIST AND THE CURRENT VERTEX TO START THE DFS

IF THE CURRENT VERTEX HAS ALREADY BEEN VISITED JUST RETURN

SET THE CURRENT VERTEX AS VISITED

FOR ALL ADJACENT VERTICES - PERFORM THE DFS

PROCESS THE NODE

UNCONNECTED GRAPH

```
// This for-loop ensures that all nodes are covered even for an unconnected
// graph.
for (int i = 0; i < N; i++) {
    depthFirstTraversal(graph, visited, i);
}
```

ITERATE THROUGH ALL NODES AND START THE DFS AT EVERY NODE TO ENSURE THAT EVEN UNCONNECTED NODES ARE COVERED

THE GRAPH TRAVERSAL

THIS IS VERY SIMILAR TO TREE TRAVERSAL

THE SAME 2 TYPES:

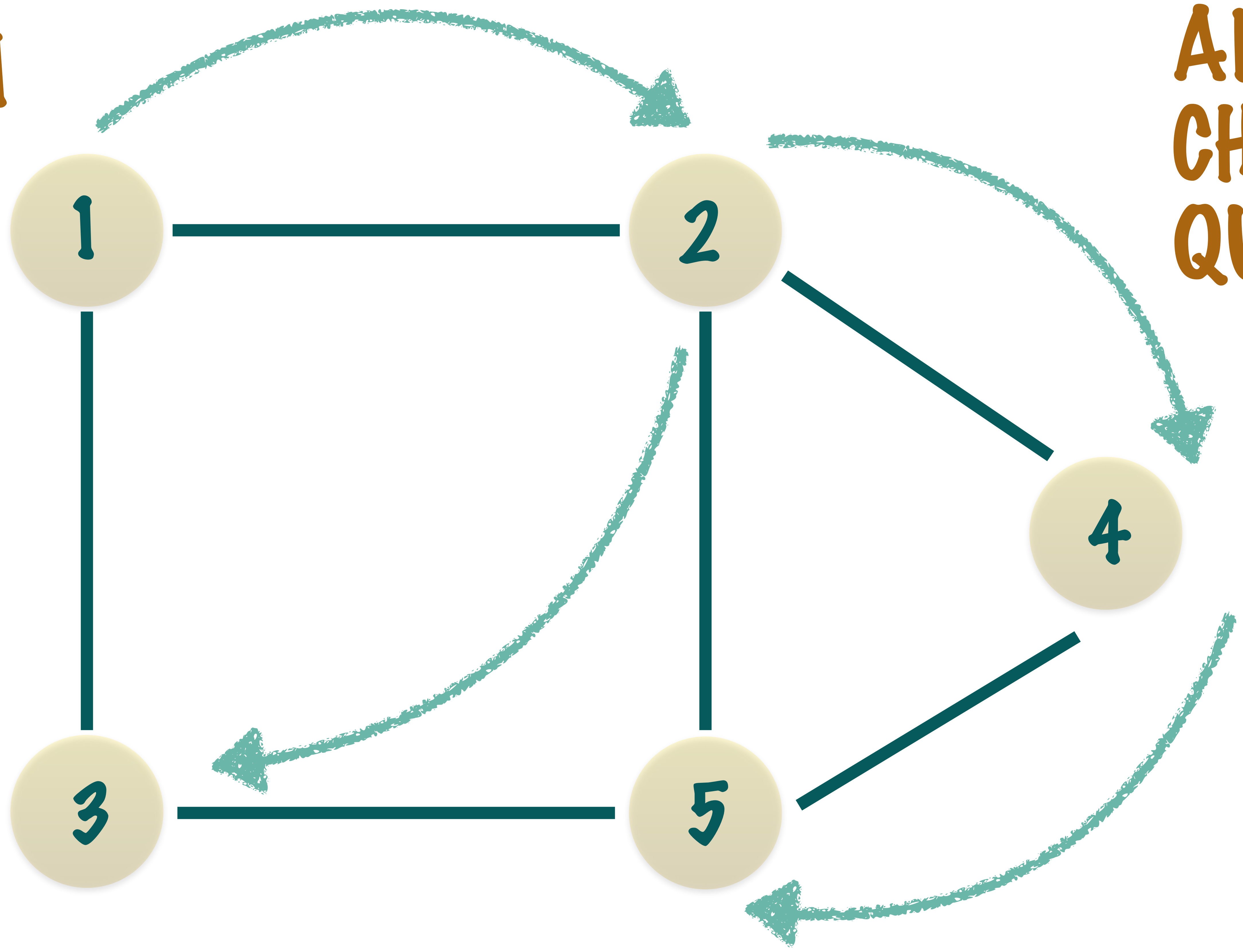
DEPTH-FIRST

BREADTH-FIRST

THE GRAPH

BREADTH-FIRST TRAVERSAL

GO LEVEL WISE FROM THE FIRST NODE



ADD NON-VISITED CHILD NODES TO A QUEUE

VISITED = TRUE MEANS THE NODE HAS BEEN SEEN BEFORE

USE A VISITED LIST TO KEEP TRACK OF NODES ALREADY VISITED

DO NOT PROCESS THOSE NODES AGAIN

THE GRAPH
BREADTH-FIRST TRAVERSAL

LET'S SEE SOME CODE...

GRAPH - BREADTH FIRST TRAVERSAL

```java
public static void breadthFirstTraversal(Graph graph, int[] visited, int currentVertex)
    throws Queue.QueueOverflowException, Queue.QueueUnderflowException {

    Queue<Integer> queue = new Queue<>(Integer.class);
    queue.enqueue(currentVertex);

    while (!queue.isEmpty()) {
        int vertex = queue.dequeue();

        if (visited[vertex] == 1) {
            continue;
        }

        System.out.print(vertex + "->");
        visited[vertex] = 1;

        List<Integer> list = graph.getAdjacentVertices(vertex);
        for (int v : list) {
            if (visited[v] != 1) {
                queue.enqueue(v);
            }
        }
    }
}
```

SPECIFIC THE GRAPH, VISITED NODE LIST AND THE CURRENT VERTEX TO START THE BFS

USE A QUEUE TO ADD THE CHILDREN IN BREADTH FIRST ORDER

CHECK IF THE NODE HAS BEEN SEEN BEFORE - IF YES CONTINUE

FOR ALL ADJACENT VERTICES - ADD IT TO THE QUEUE TO VISIT IN BFS FORM

PROCESS AND VISIT THE NODE

UNCONNECTED GRAPH

```
// This for-loop ensures that all nodes are covered even for an unconnected
// graph.
for (int i = 0; i < N; i++) {
    breadthFirstTraversal(graph, visited, i);
}
```

ITERATE THROUGH ALL NODES AND START THE BFS AT EVERY NODE TO ENSURE THAT EVEN UNCONNECTED NODES ARE COVERED