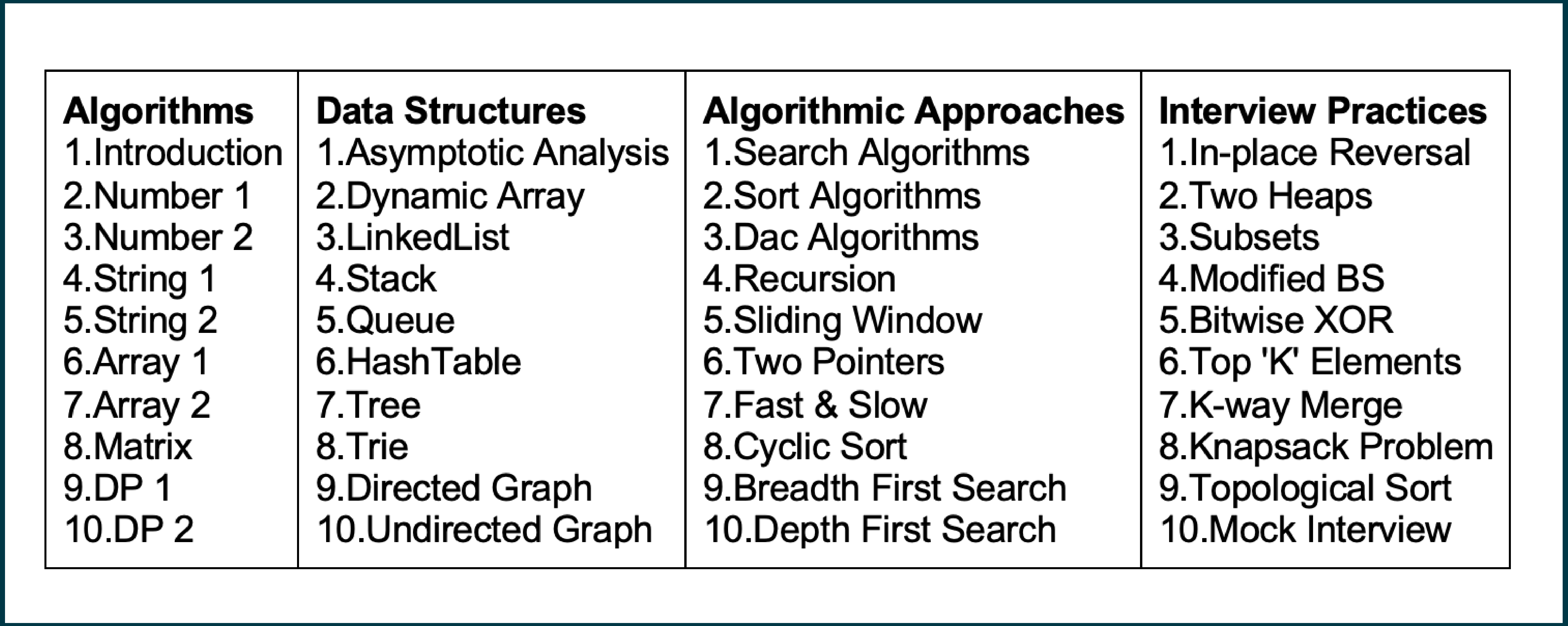
PDP ONLINE

DSA – Data Structures

Directed Graph

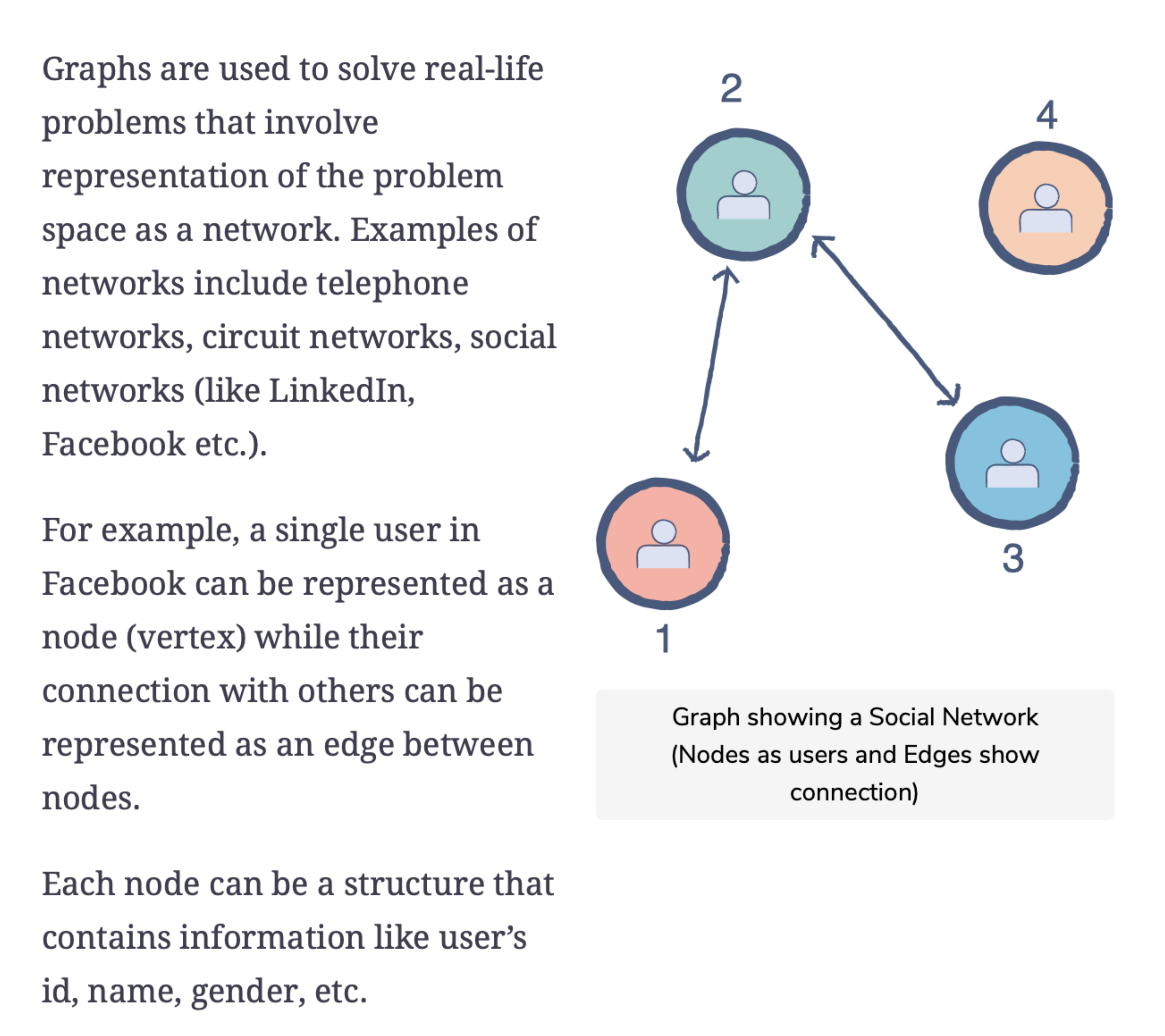
Course Planning



Algorithms	Data Structures	Algorithmic Approaches	Interview Practices
1.Introduction	1.Asymptotic Analysis	1.Search Algorithms	1.In-place Reversal
2.Number 1	2.Dynamic Array	2.Sort Algorithms	2.Two Heaps
3.Number 2	3.LinkedList	3.Dac Algorithms	3.Subsets
4.String 1	4.Stack	4.Recursion	4.Modified BS
5.String 2	5.Queue	5.Sliding Window	5.Bitwise XOR
6.Array 1	6.HashTable	6.Two Pointers	6.Top 'K' Elements
7.Array 2	7.Tree	7.Fast & Slow	7.K-way Merge
8.Matrix	8.Trie	8.Cyclic Sort	8.Knapsack Problem
9.DP 1	9.Directed Graph	9.Breadth First Search	9.Topological Sort
10.DP 2	10.Undirected Graph	10.Depth First Search	10.Mock Interview

https://kurbanovxurshid.medium.com/data-structures-and-algorithms-course-plan-1ab912ec1e17

What are Graphs?



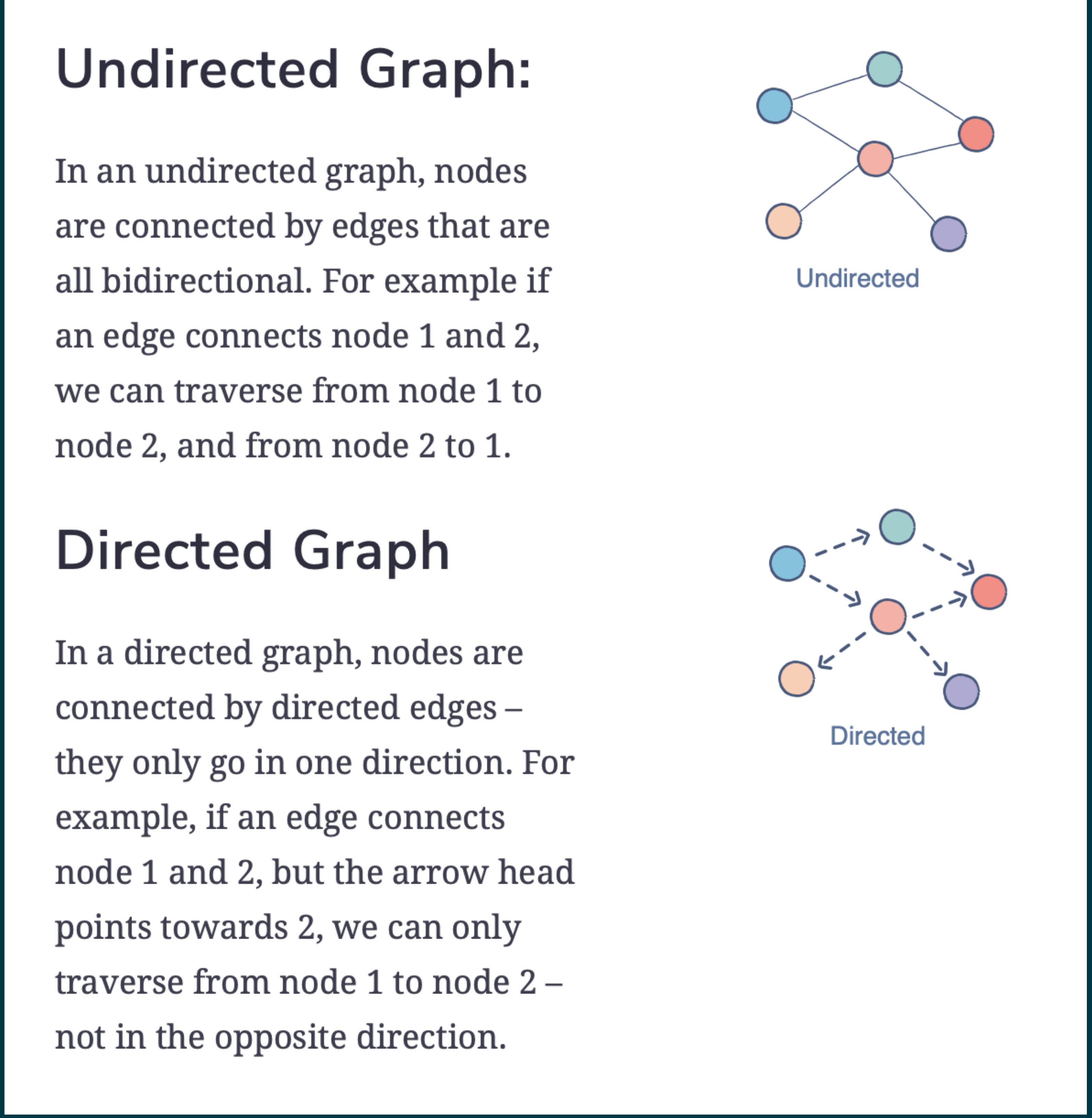
Graphs are used to solve real-life problems that involve representation of the problem space as a network. Examples of networks include telephone networks, circuit networks, social networks (like LinkedIn, Facebook etc.).

For example, a single user in Facebook can be represented as a node (vertex) while their connection with others can be represented as an edge between nodes.

Each node can be a structure that contains information like user's id, name, gender, etc.

Graph showing a Social Network (Nodes as users and Edges show connection)

What are Graphs?



Undirected Graph:

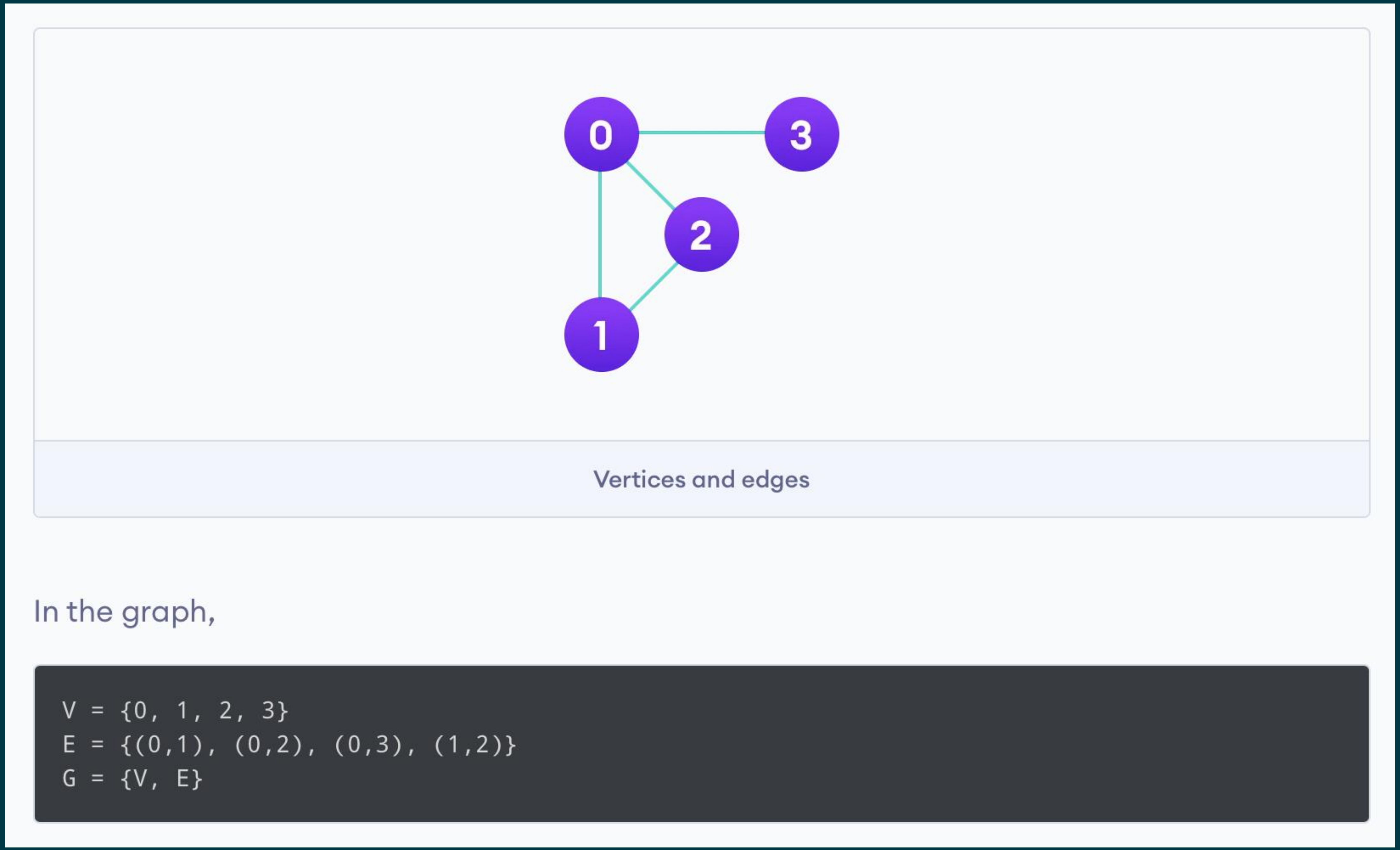
In an undirected graph, nodes are connected by edges that are all bidirectional. For example if an edge connects node 1 and 2, we can traverse from node 1 to node 2, and from node 2 to 1.

Undirected

Directed Graph

In a directed graph, nodes are connected by directed edges – they only go in one direction. For example, if an edge connects node 1 and 2, but the arrow head points towards 2, we can only traverse from node 1 to node 2 – not in the opposite direction.

Vertices & Edges

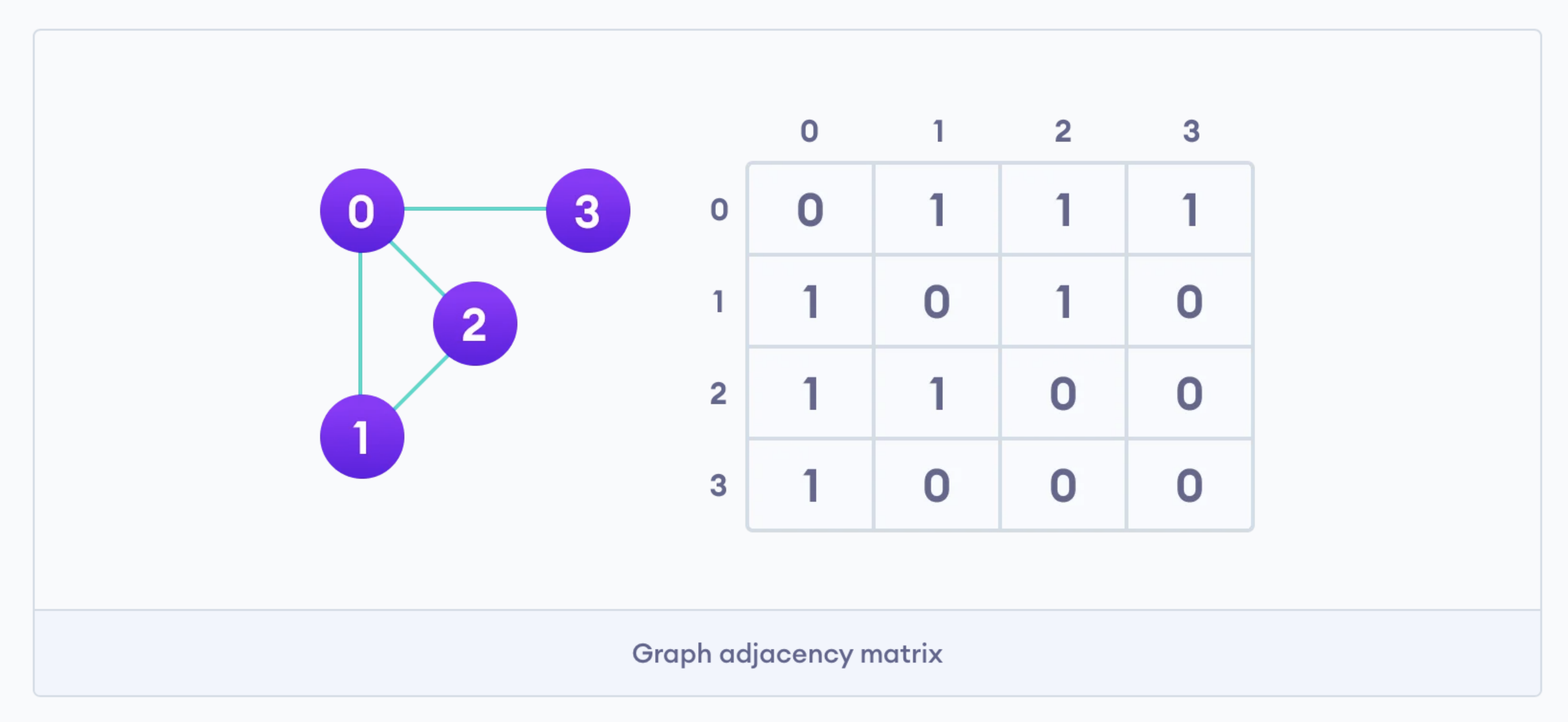


Vertices and edges

In the graph,

```
V = {0, 1, 2, 3}
E = {(0,1), (0,2), (0,3), (1,2)}
G = {V, E}
```

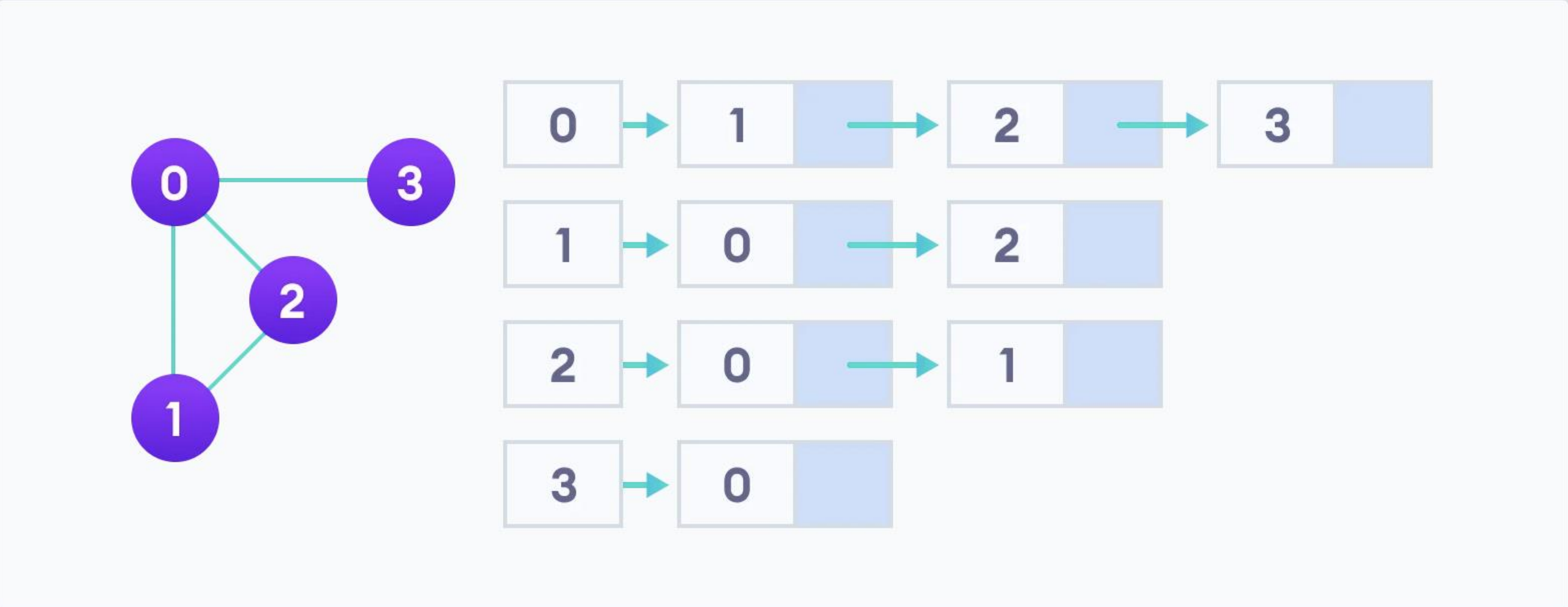
Adjacency Matrix



$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 & 0 \\ 3 & 1 & 0 & 0 & 0 \end{array}$$

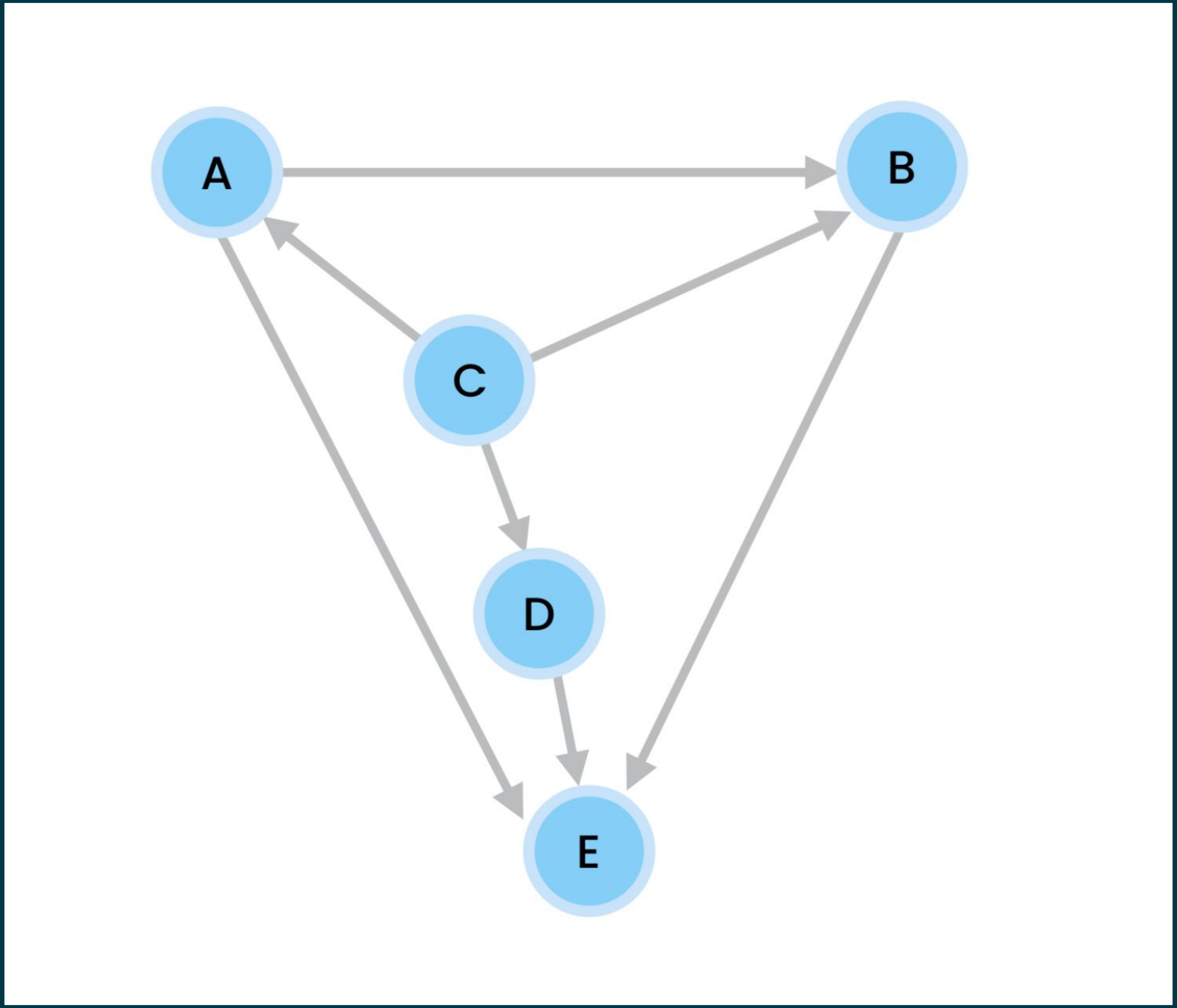
Graph adjacency matrix

Adjacency List

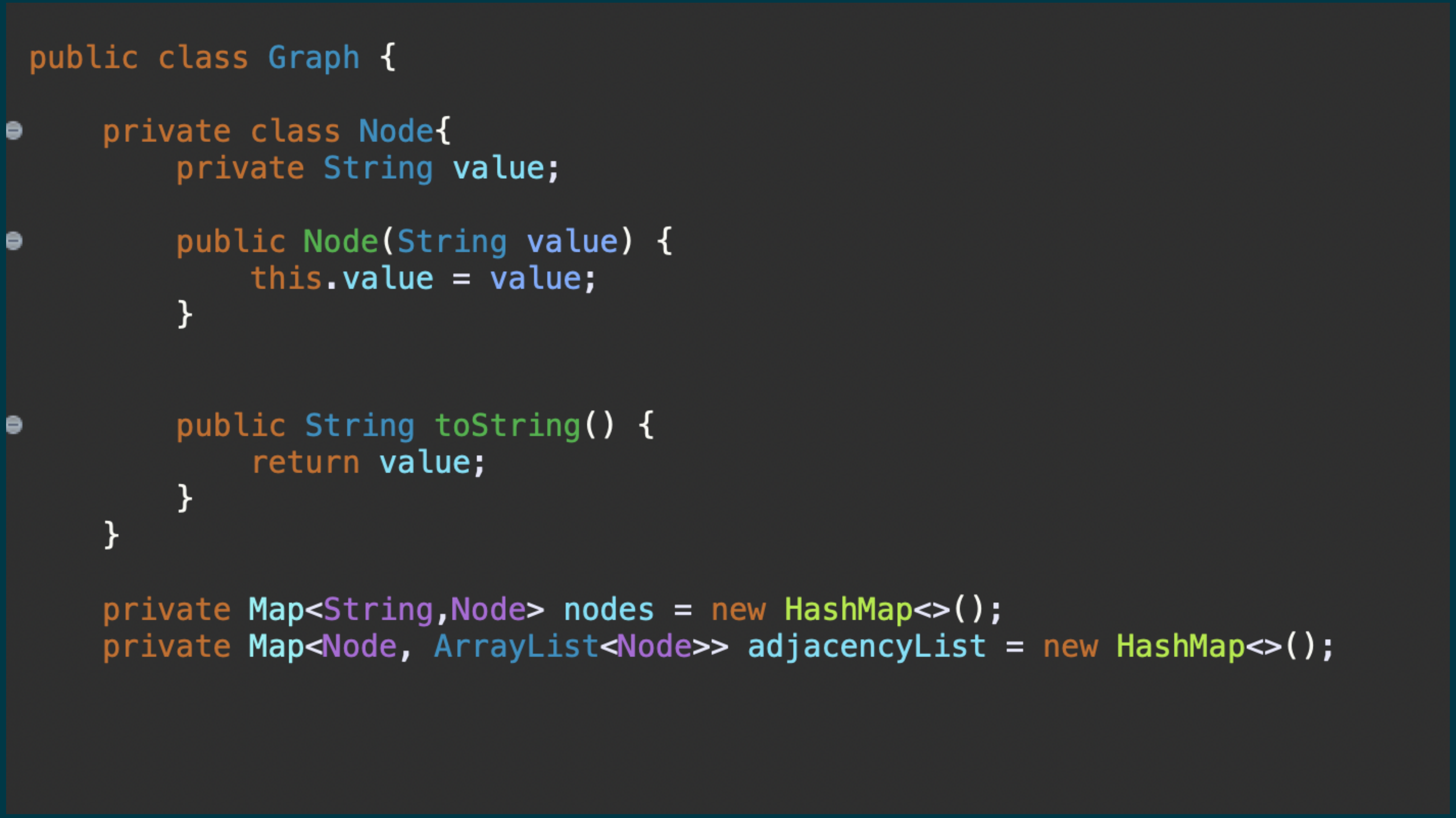


Adjacency list representation

Directed Graph

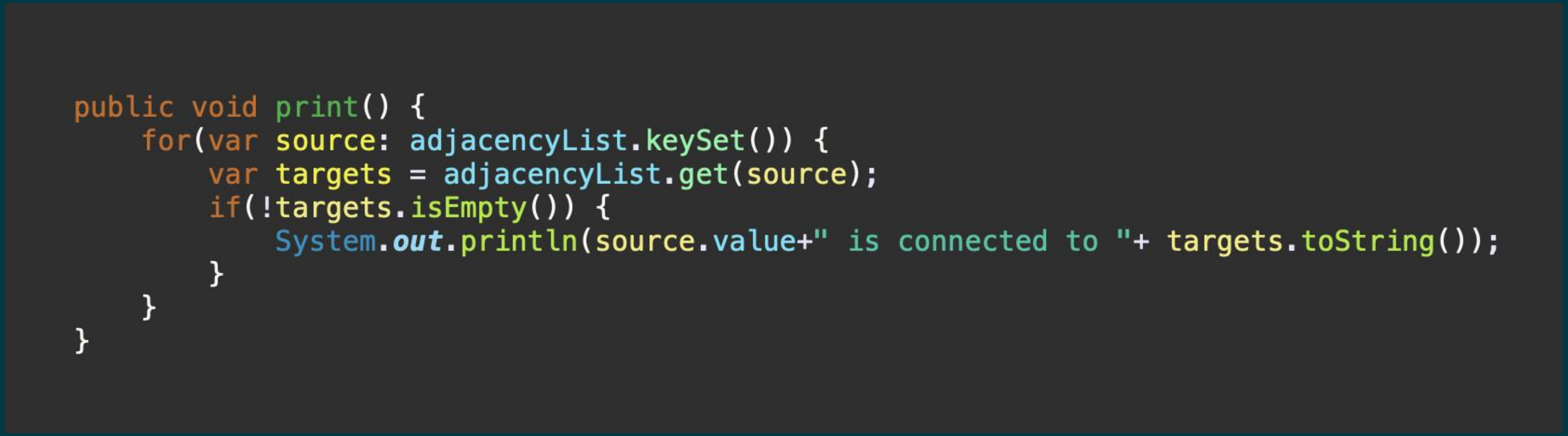


Create Graph



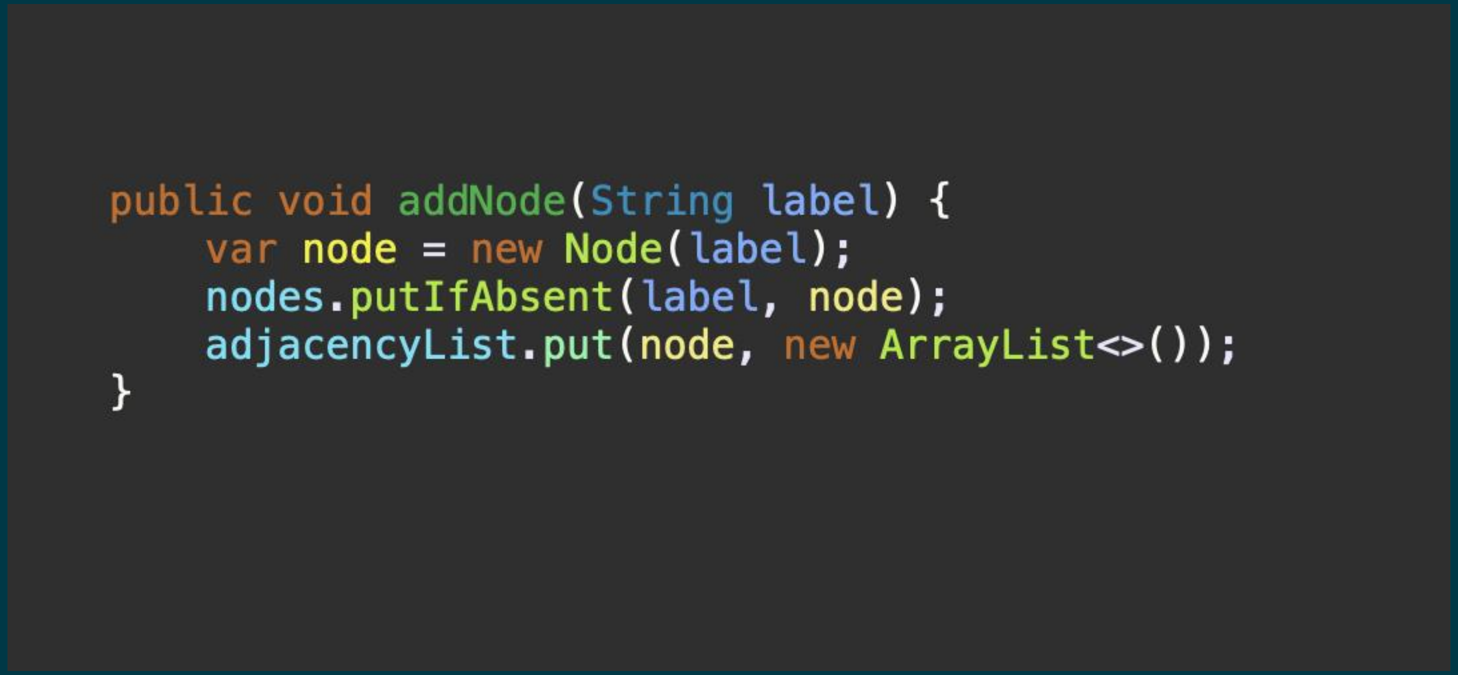
```
public class Graph {

    private class Node{
        private String value;

        public Node(String value) {
            this.value = value;
        }


        public String toString() {
            return value;
        }
    }

    private Map<String,Node> nodes = new HashMap<>();
    private Map<Node, ArrayList<Node>> adjacencyList = new HashMap<>();
```

Print



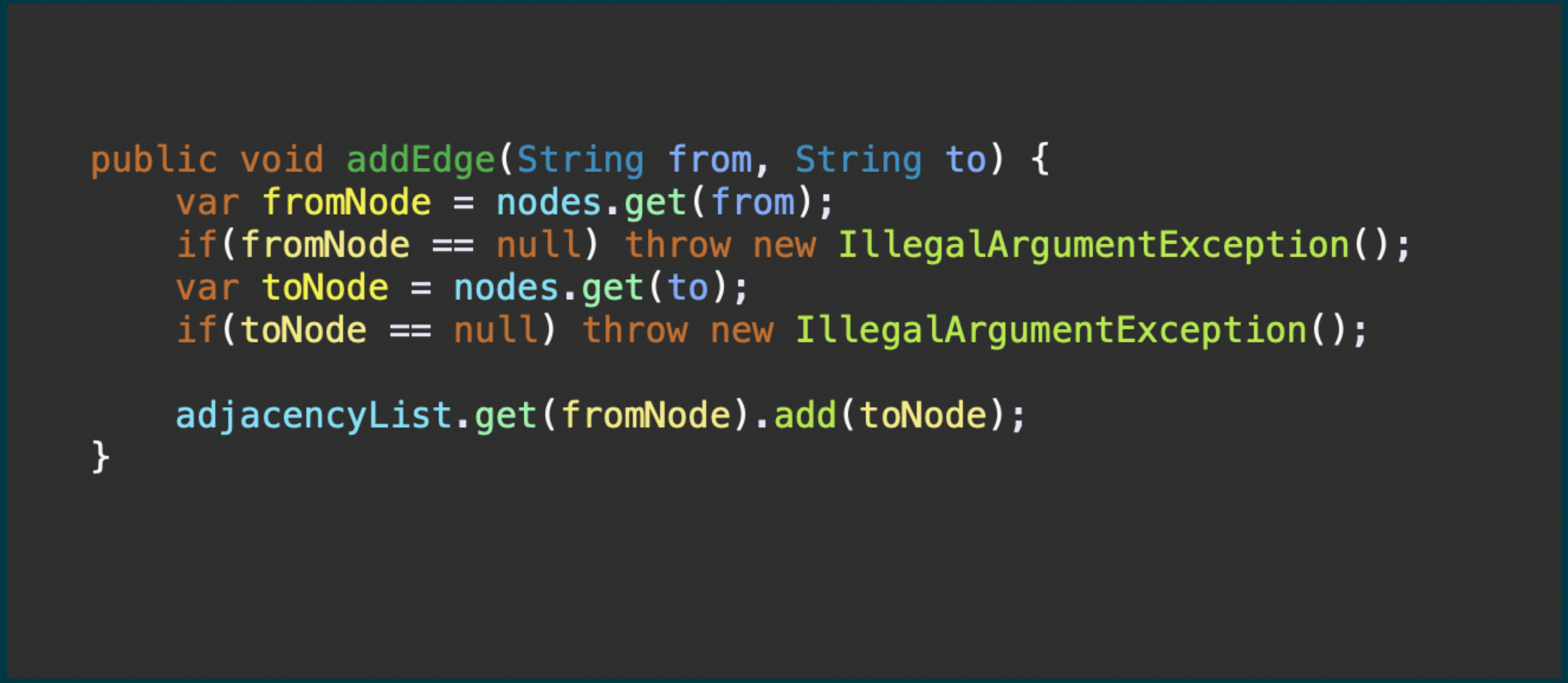
```java
public void print() {
    for(var source: adjacencyList.keySet()) {
        var targets = adjacencyList.get(source);
        if(!targets.isEmpty()) {
            System.out.println(source.value+" is connected to "+ targets.toString());
        }
    }
}
```

Add Node



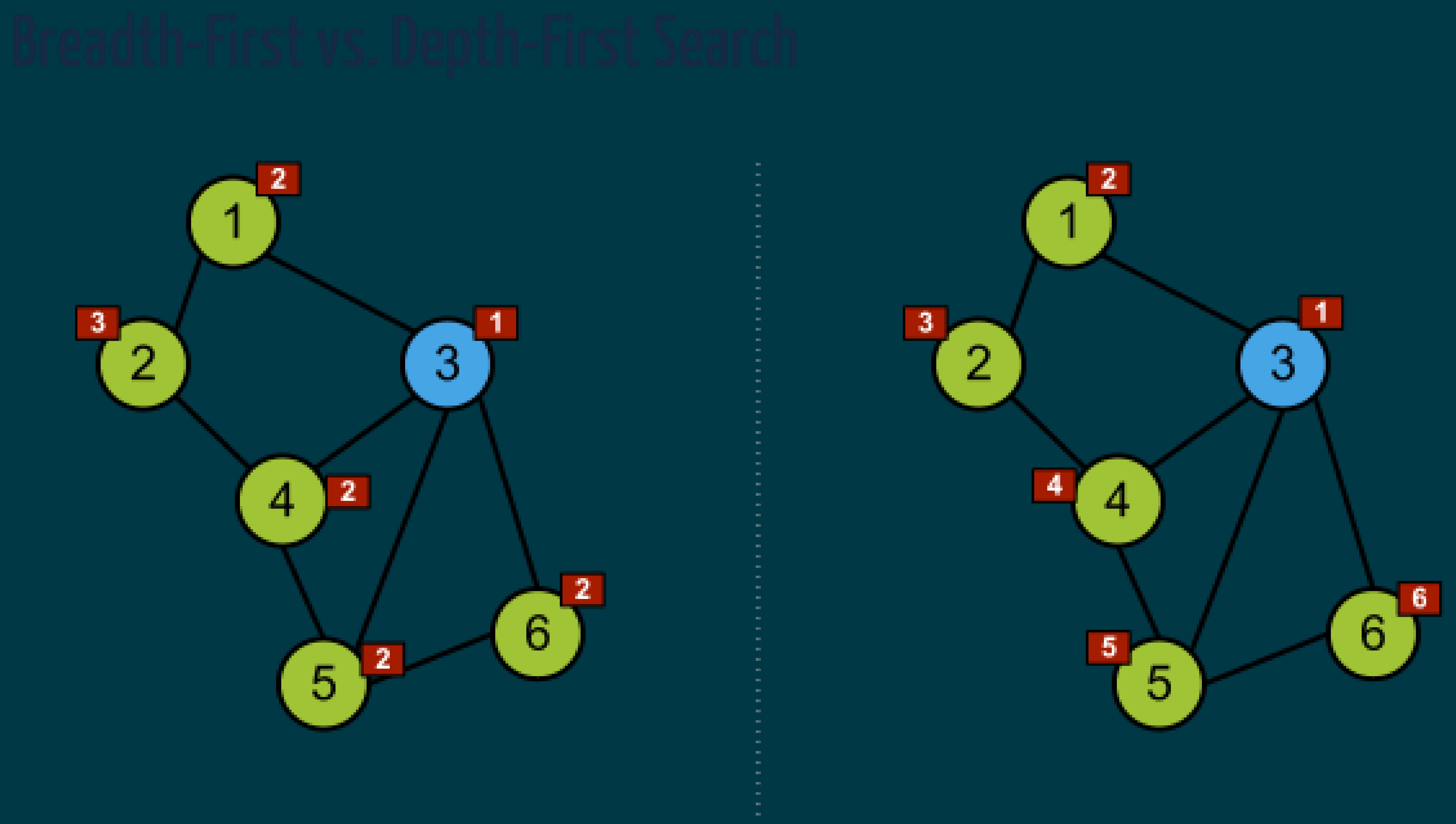
```
public void addNode(String label) {
    var node = new Node(label);
    nodes.putIfAbsent(label, node);
    adjacencyList.put(node, new ArrayList<>());
}
```

Add Edge

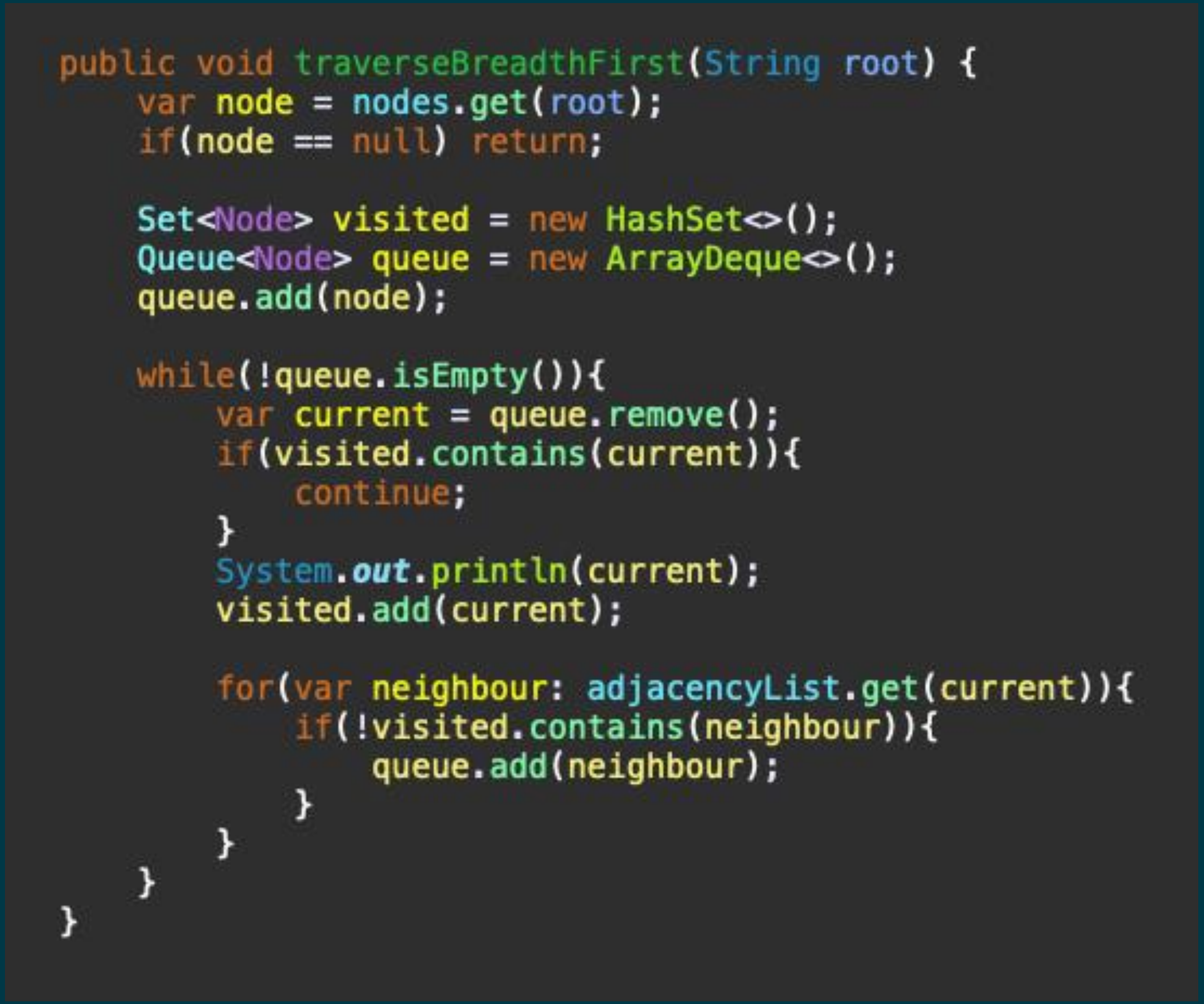


```java
public void addEdge(String from, String to) {
    var fromNode = nodes.get(from);
    if(fromNode == null) throw new IllegalArgumentException();
    var toNode = nodes.get(to);
    if(toNode == null) throw new IllegalArgumentException();

    adjacencyList.get(fromNode).add(toNode);
}
```

Breadth First & Depth First

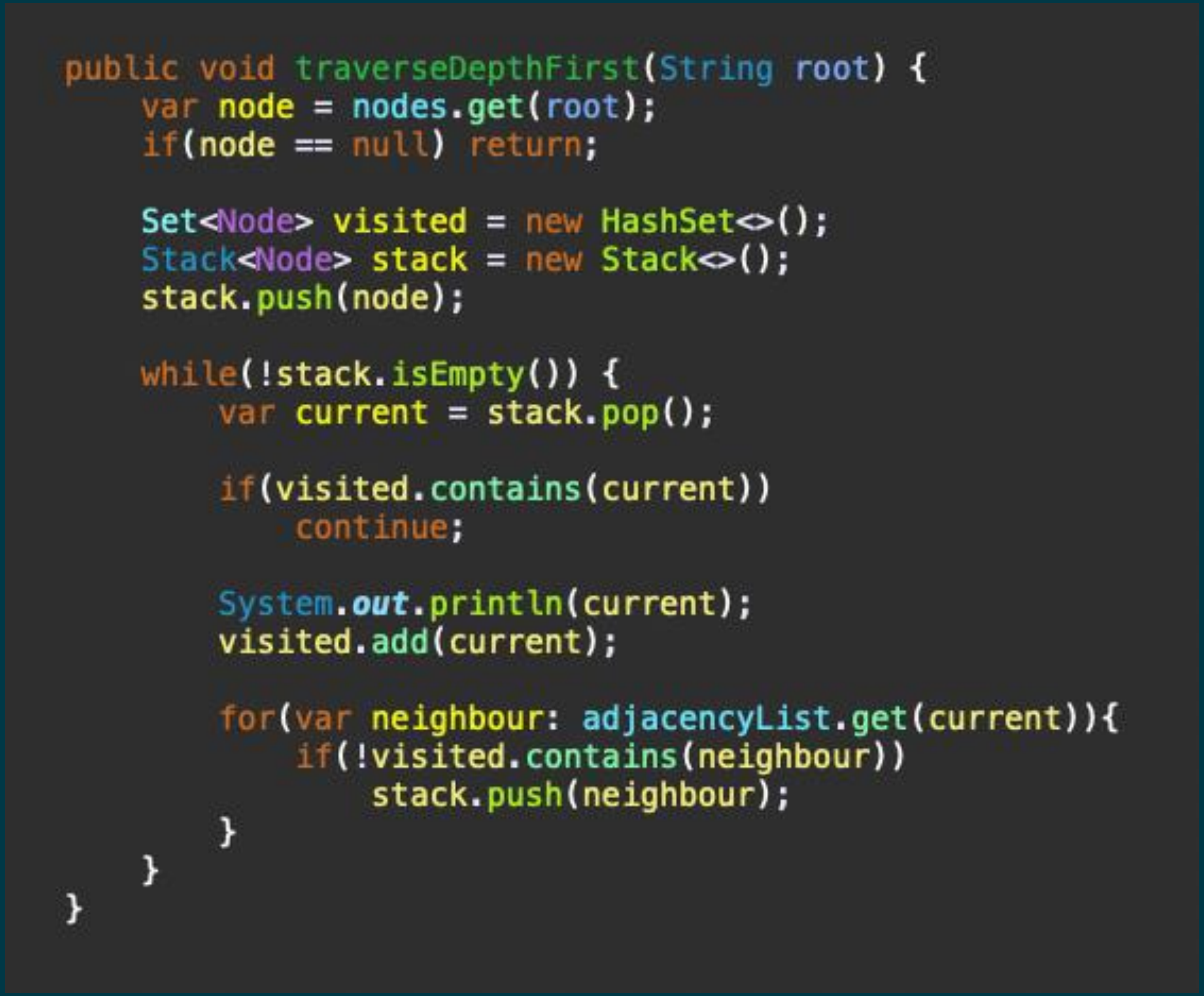


Traverse Breadth First



```java
public void traverseBreadthFirst(String root) {
    var node = nodes.get(root);
    if(node == null) return;

    Set<Node> visited = new HashSet<>();
    Queue<Node> queue = new ArrayDeque<>();
    queue.add(node);

    while(!queue.isEmpty()){
        var current = queue.remove();
        if(visited.contains(current)){
            continue;
        }
        System.out.println(current);
        visited.add(current);

        for(var neighbour: adjacencyList.get(current)){
            if(!visited.contains(neighbour)){
                queue.add(neighbour);
            }
        }
    }
}
```

Traverse Depth First



```java
public void traverseDepthFirst(String root) {
    var node = nodes.get(root);
    if(node == null) return;

    Set<Node> visited = new HashSet<>();
    Stack<Node> stack = new Stack<>();
    stack.push(node);

    while(!stack.isEmpty()) {
        var current = stack.pop();

        if(visited.contains(current))
            continue;

        System.out.println(current);
        visited.add(current);

        for(var neighbour: adjacencyList.get(current)){
            if(!visited.contains(neighbour))
                stack.push(neighbour);
        }
    }
}
```

Task 1

Darsda o`tilgan Graph da Node ni o`chirish imkoniyatini yarating.

```
public void removeNode(String label)
```

Task 2

Darsda o`tilgan Graph da Edge ni o`chirish imkoniyatini yarating.

```
public void removeNode(String from, String to)
```