MATH-810 Mathematical Methods for Artificial Intelligence

DFS & BFS

Dr. Yasir Ali

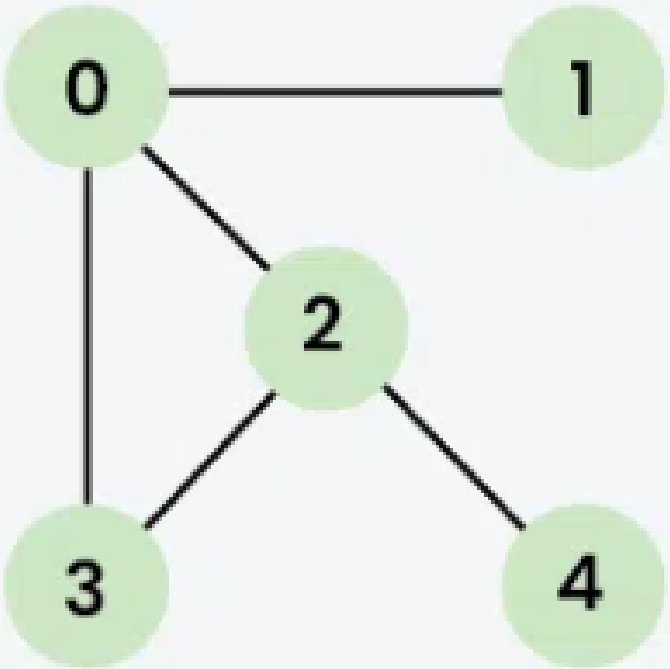
DBS&H, CEME
National University of Sciences and Technology
Islamabad, Pakistan.

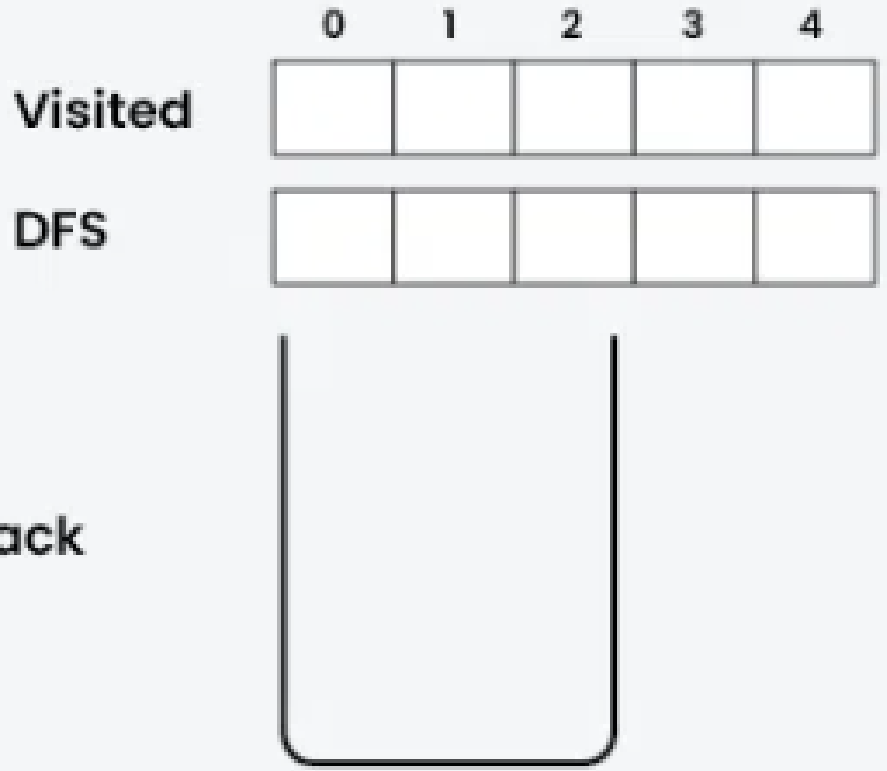
September 24, 2025

Depth-First Search (DFS)

1. Start at a given node.
2. Mark the current node as visited.
3. Explore each unvisited neighbor recursively before backtracking.
4. Continue until all reachable nodes are visited.

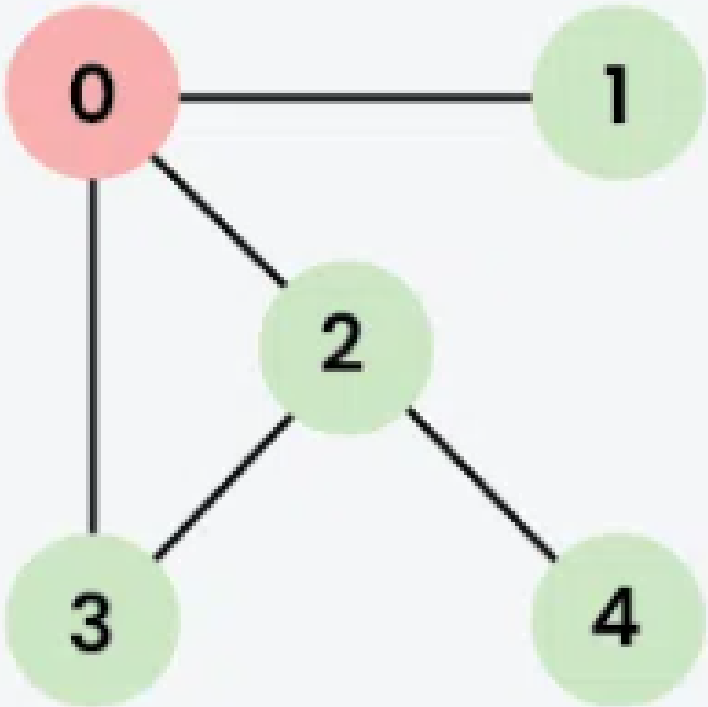
Initially | Initially stack and visited arrays are empty.





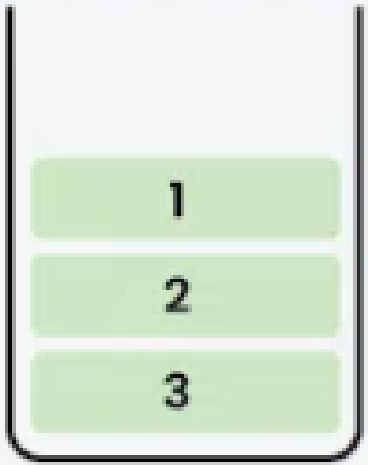
DFS on Graph

01 Step | Visit 0 and put its adjacent nodes which are not visited yet into the stack.



	0	1	2	3	4
Visited	T	T	T	T	
DFS	0				

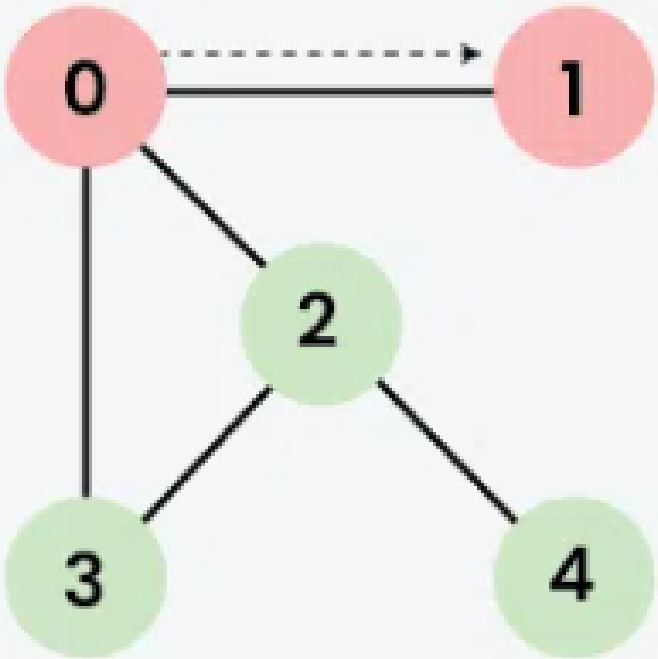
Stack



DFS on Graph

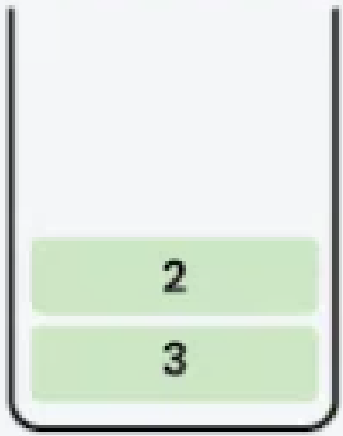
02 Step

Now, Node 1 at the top of the stack, so visit node 1 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



	0	1	2	3	4
Visited	T	T	T	T	
DFS	0	1			

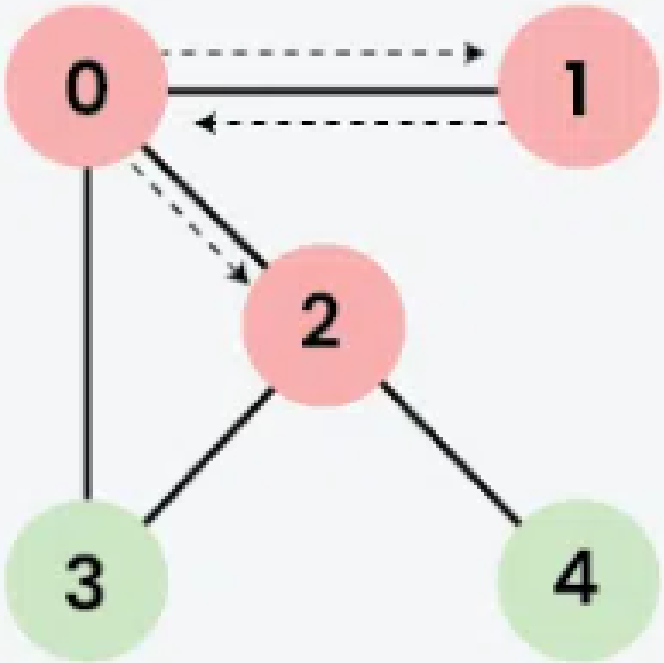
Stack



DFS on Graph

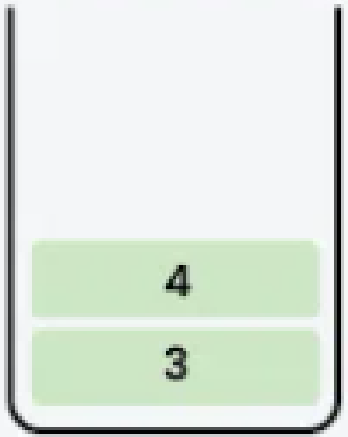
03 Step

Now, Node 2 at the top of the stack, so visit node 2 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



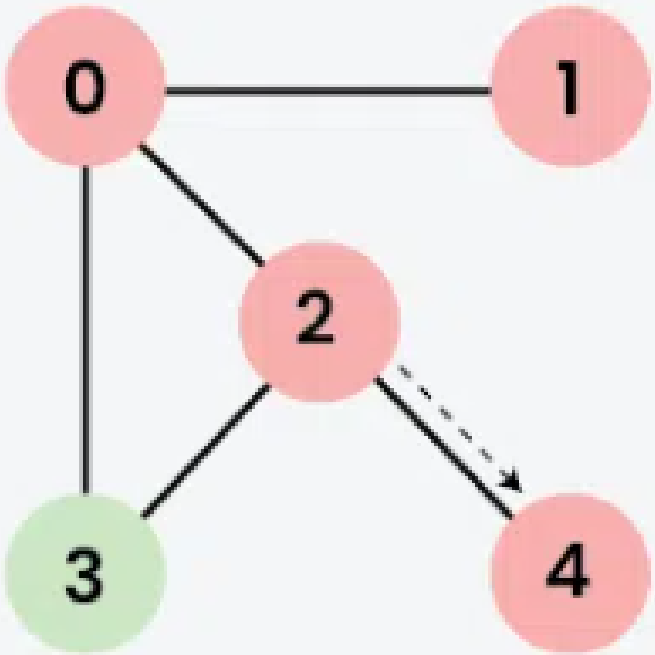
	0	1	2	3	4
Visited	T	T	T	T	T
DFS	0	1	2		

Stack



DFS on Graph

04 Step | Now, Node 4 at the top of the stack, so visit node 4 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



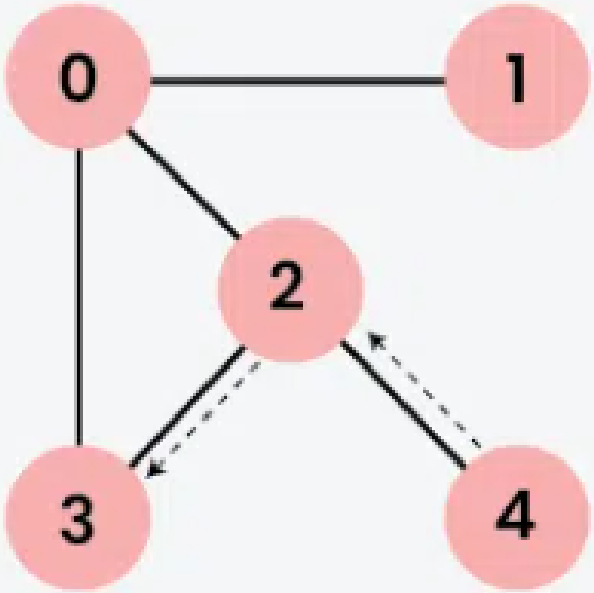
	0	1	2	3	4
Visited	T	T	T	T	T
DFS	0	1	2	4	

Stack

DFS on Graph

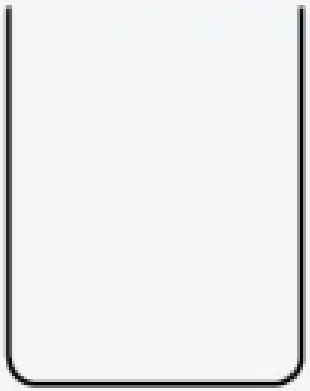
05 Step

Now, Node 3 at the top of the stack, so visit node 3 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



	0	1	2	3	4
Visited	T	T	T	T	T
DFS	0	1	2	4	3

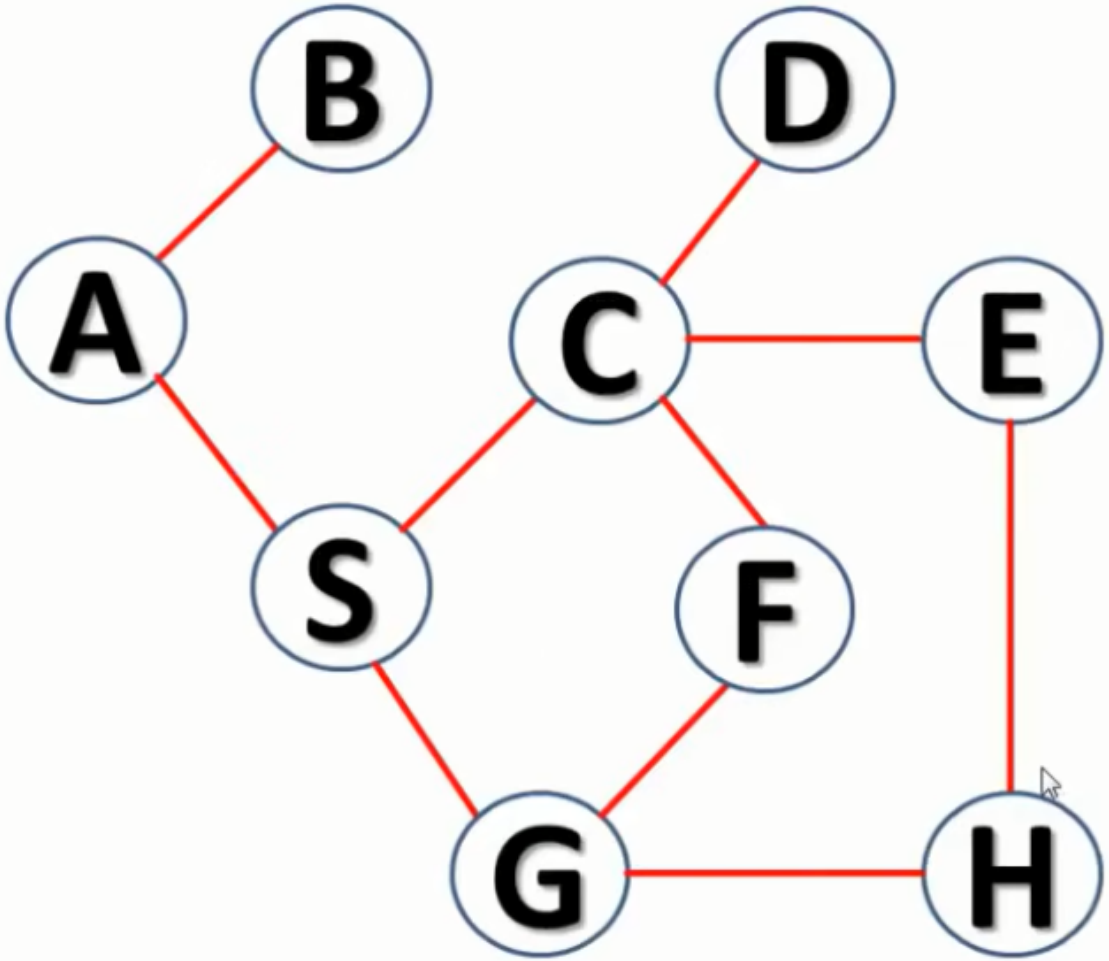
Stack



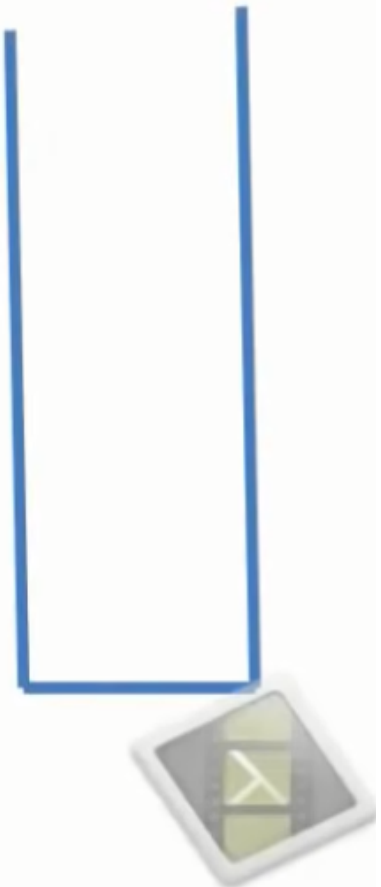
Now, Stack becomes empty, which means we have visited all the nodes and our DFS traversal ends.

DFS on Graph

DEPTH FIRST SEARCH

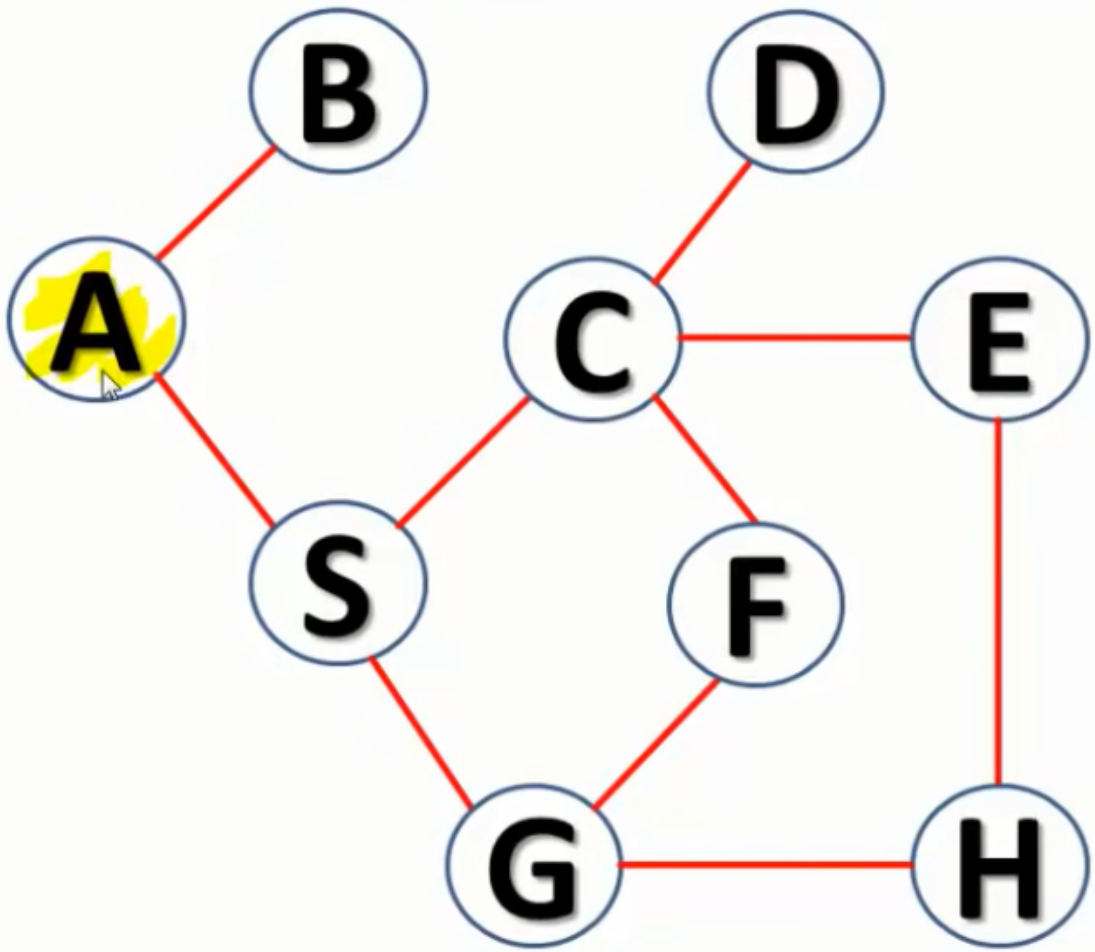


Stack Status

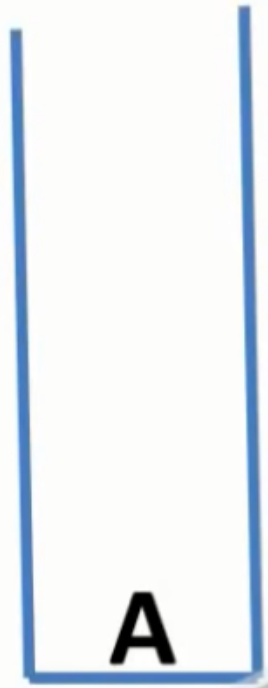


OUTPUT :

DEPTH FIRST SEARCH

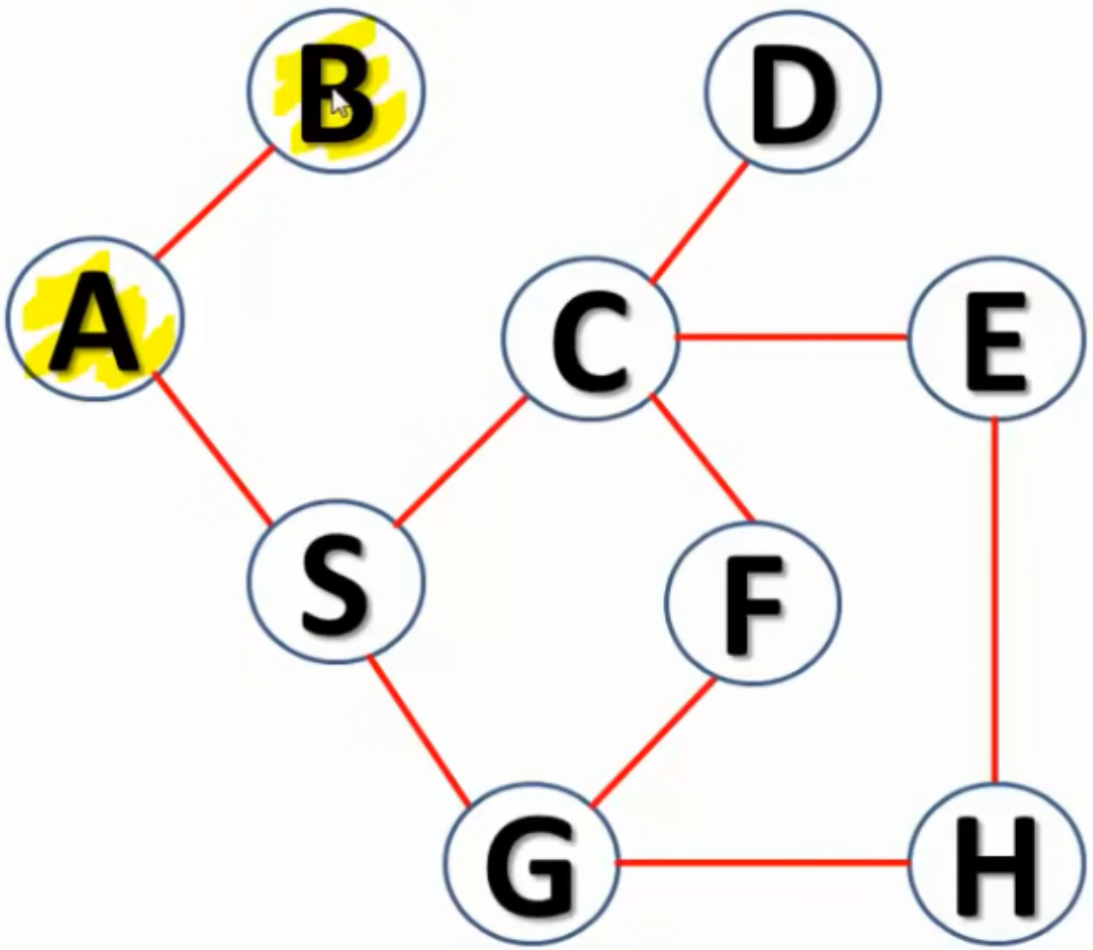


Stack Status

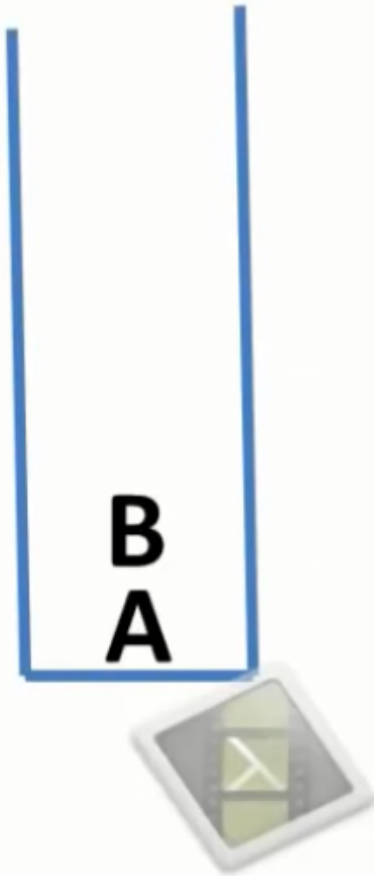


OUTPUT : A

DEPTH FIRST SEARCH

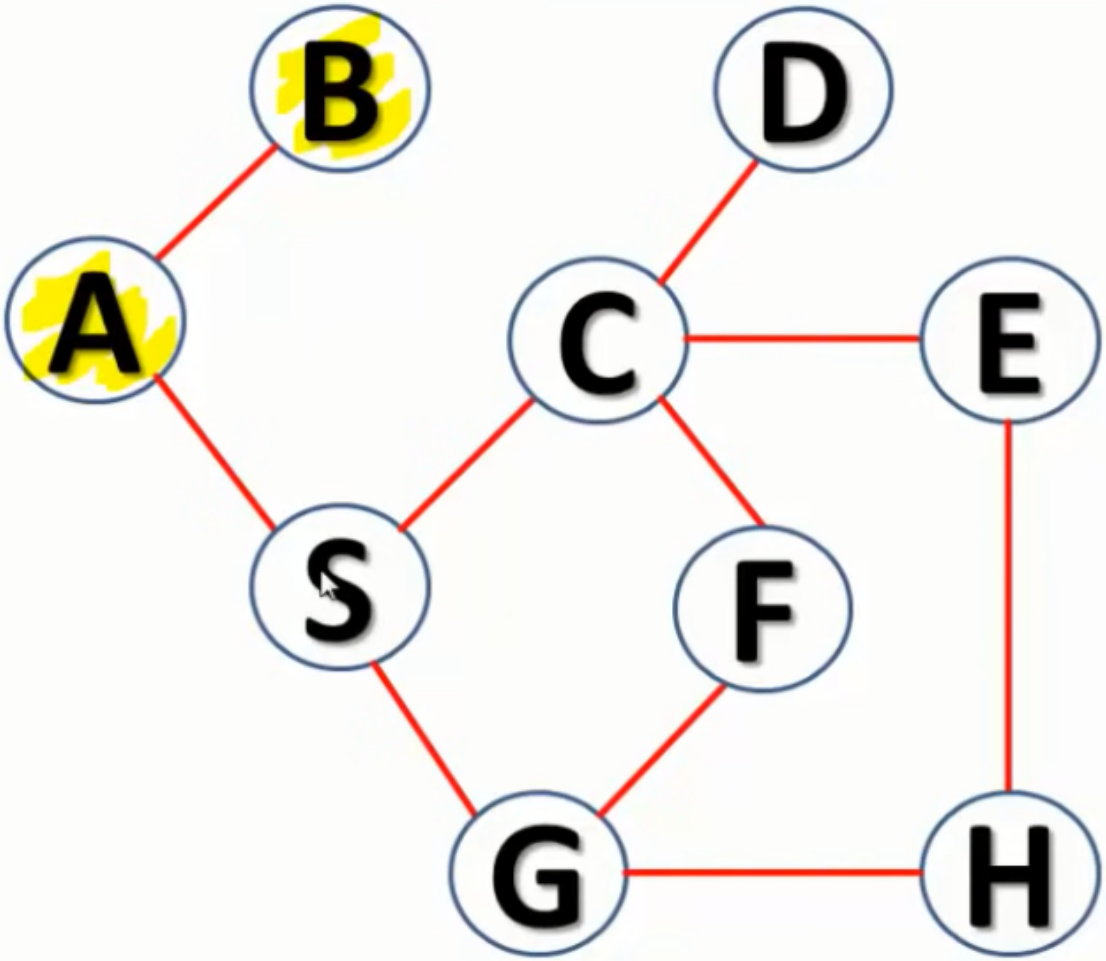


Stack Status



OUTPUT : A B

DEPTH FIRST SEARCH

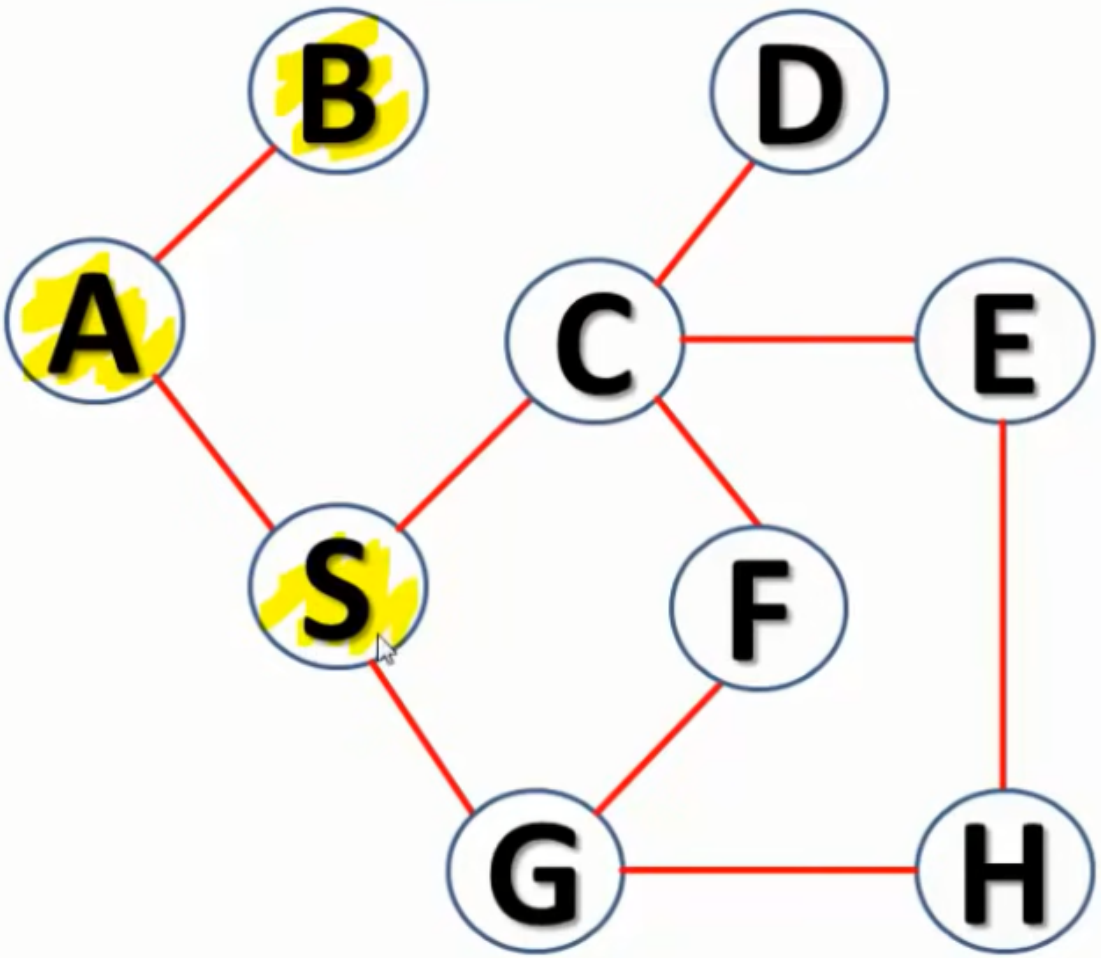


Stack Status

B is removed ferom stack there is no unvisited vertex from B

OUTPUT : A B

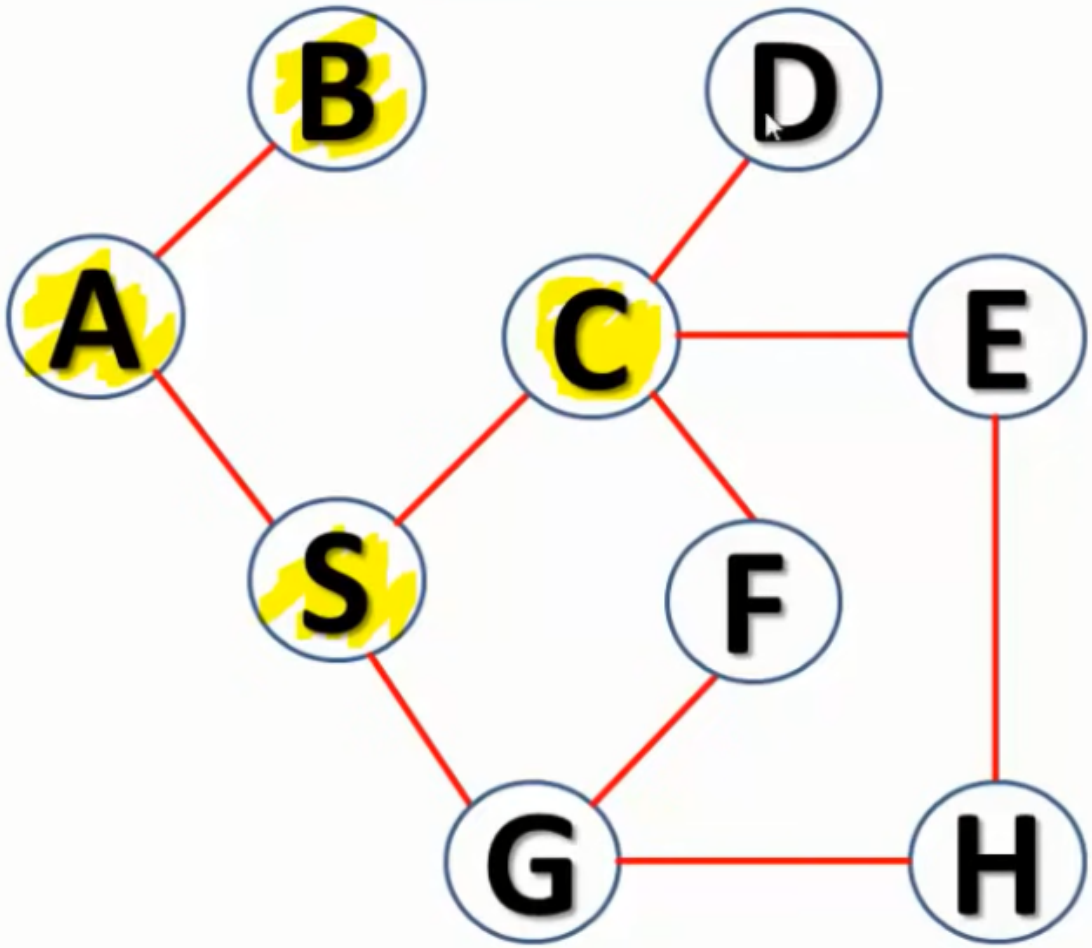
DEPTH FIRST SEARCH



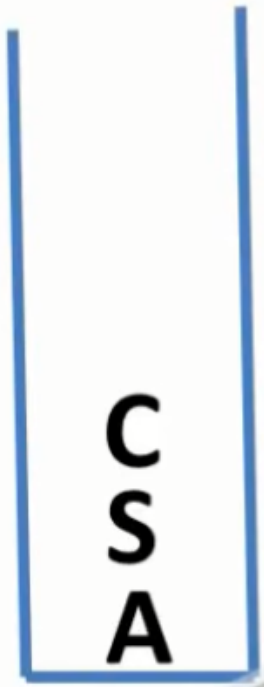
Stack Status

OUTPUT : A B S

DEPTH FIRST SEARCH

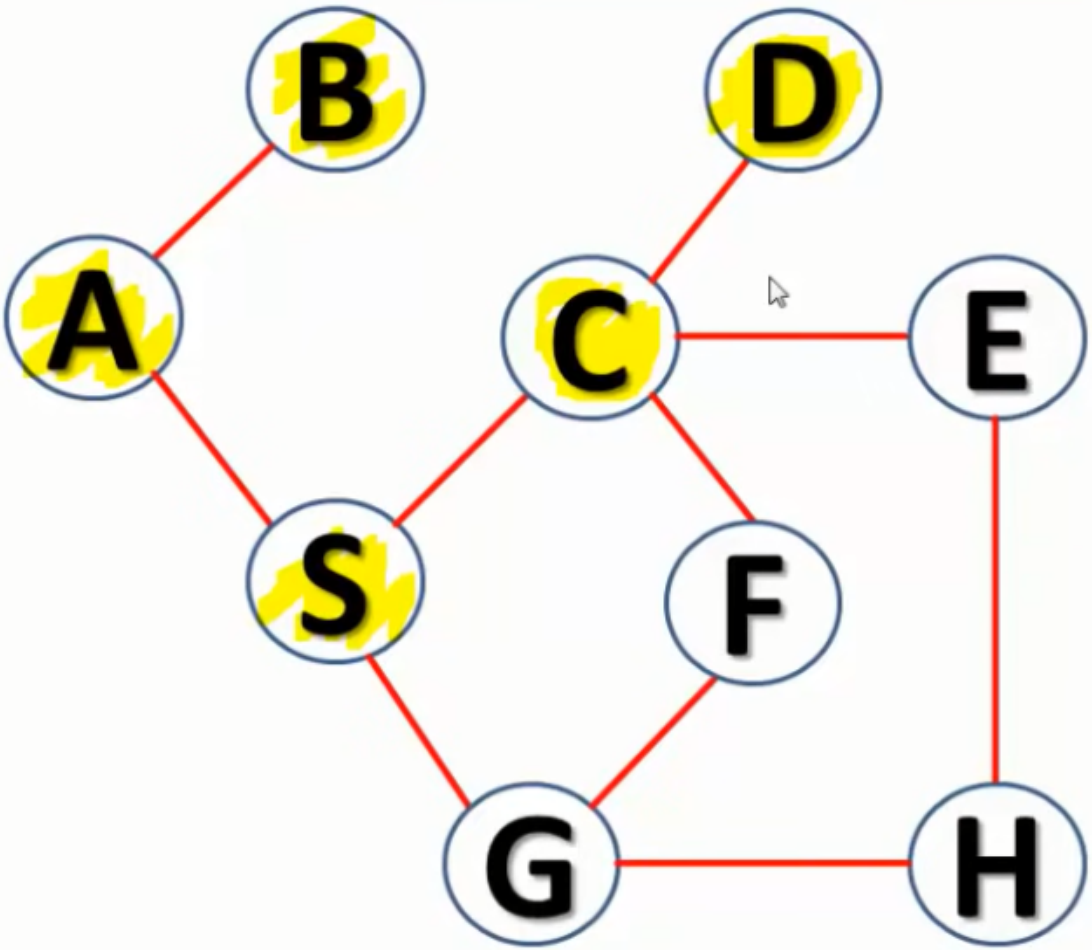


Stack Status



OUTPUT : A B S C

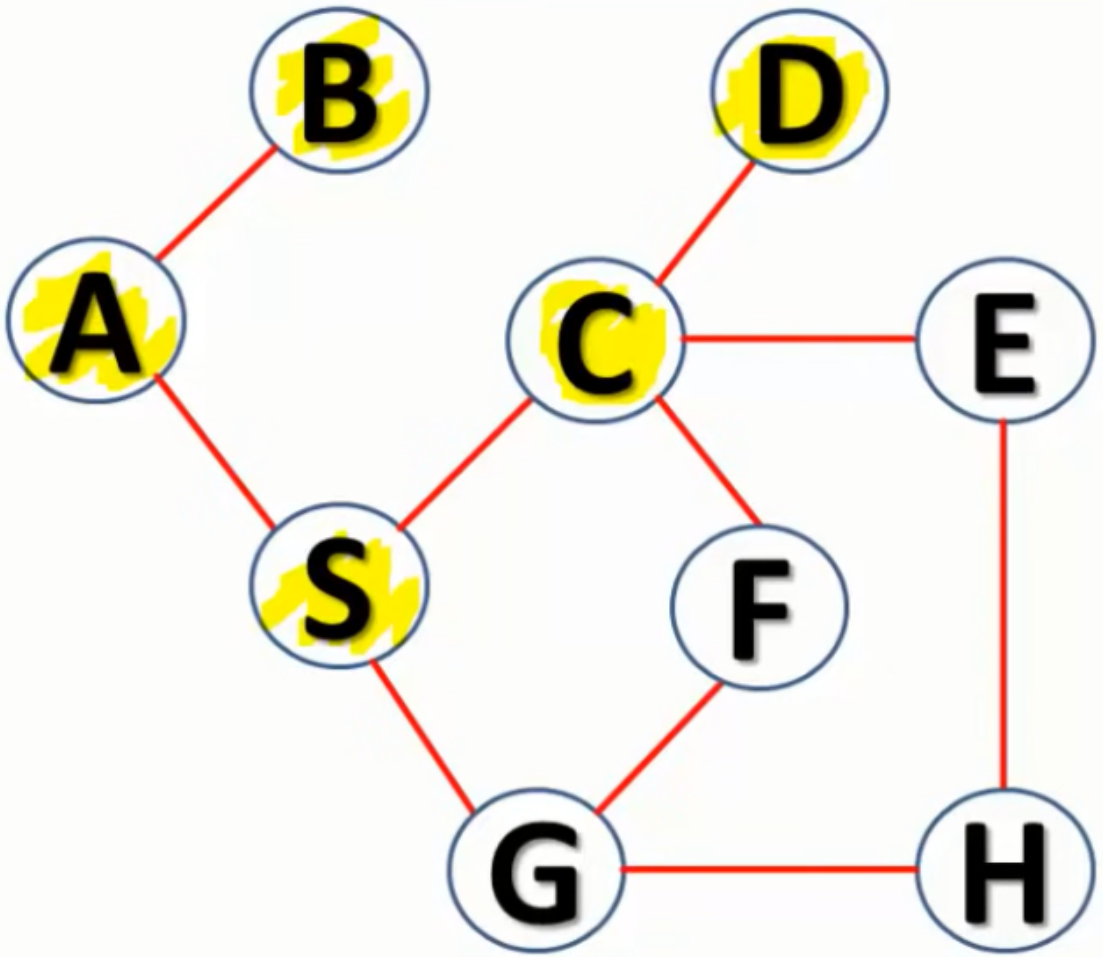
DEPTH FIRST SEARCH



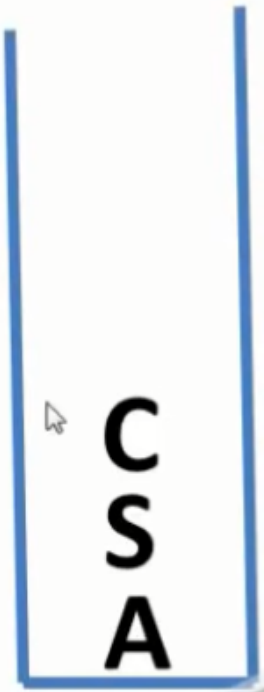
Stack Status

OUTPUT : A B S C D

DEPTH FIRST SEARCH

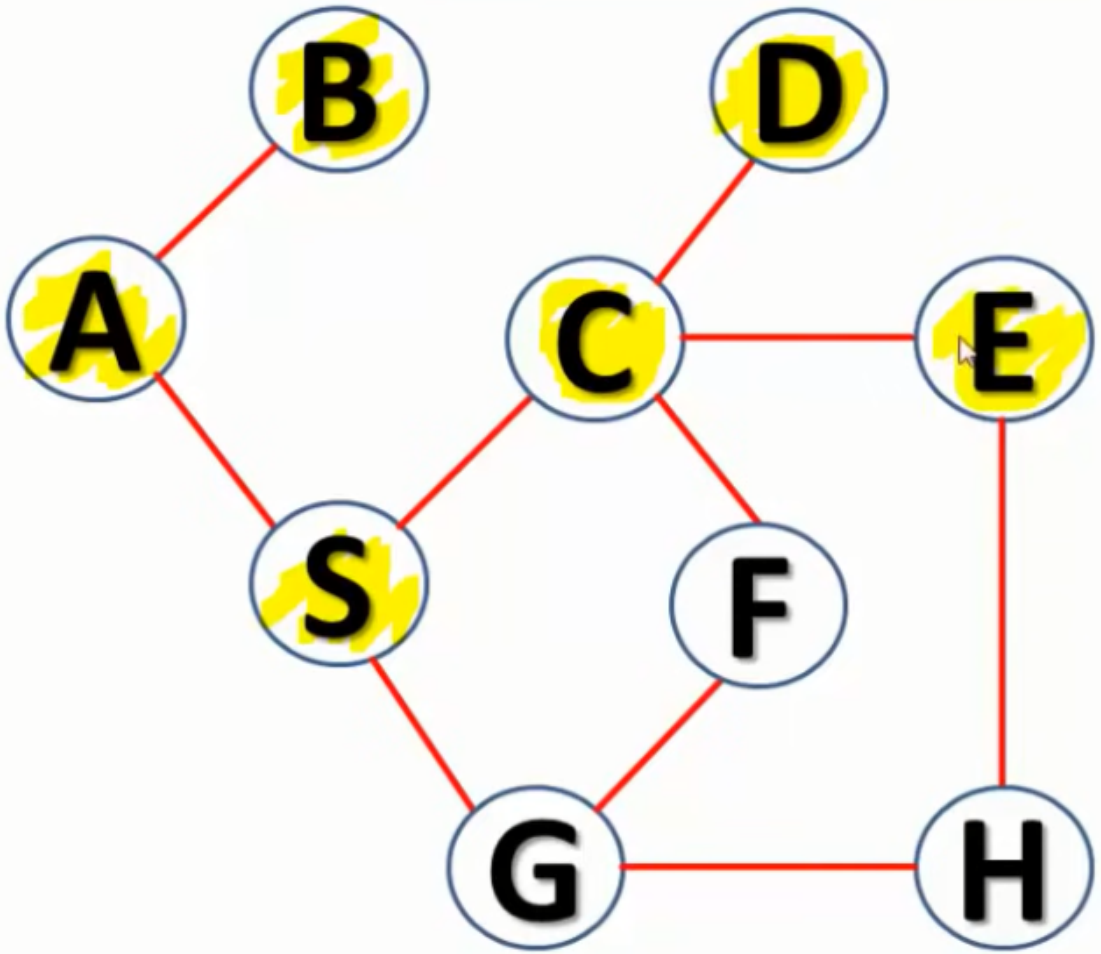


Stack Status



OUTPUT : A B S C D

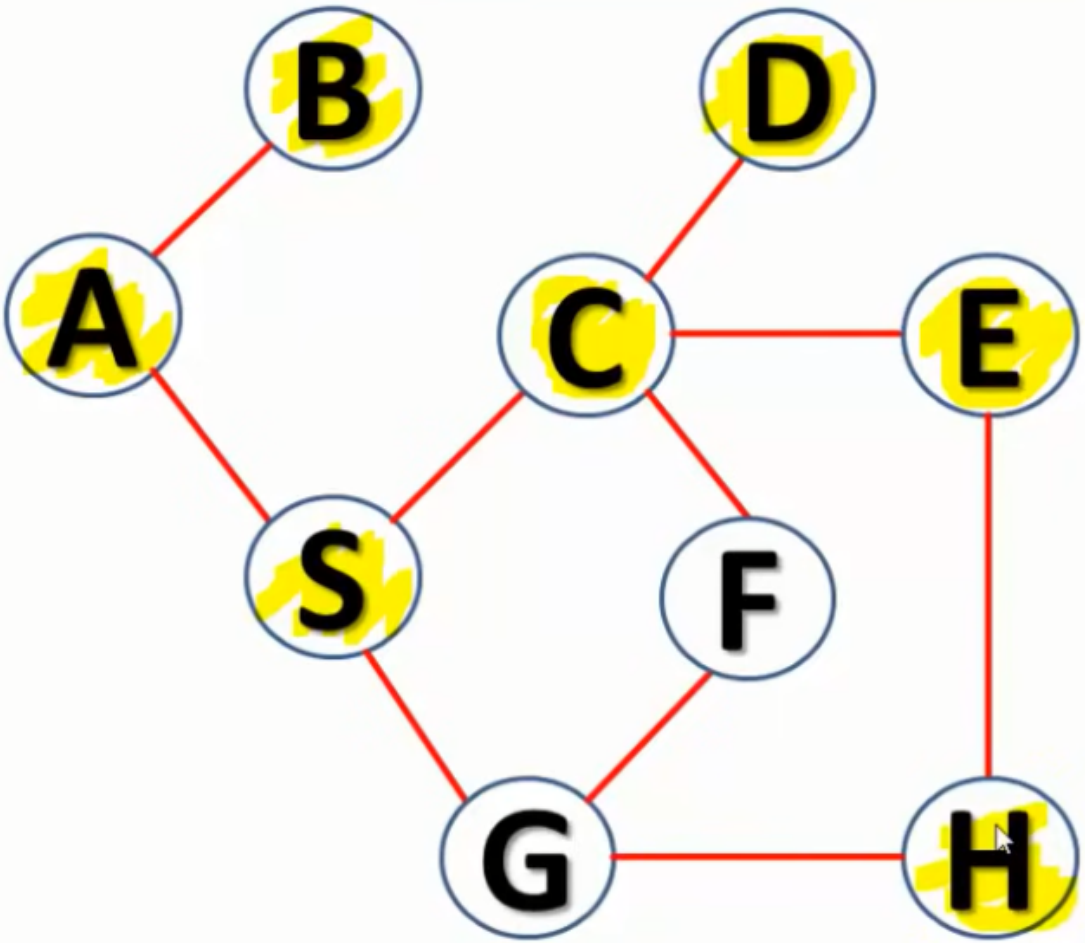
DEPTH FIRST SEARCH



Stack Status

OUTPUT : A B S C D E

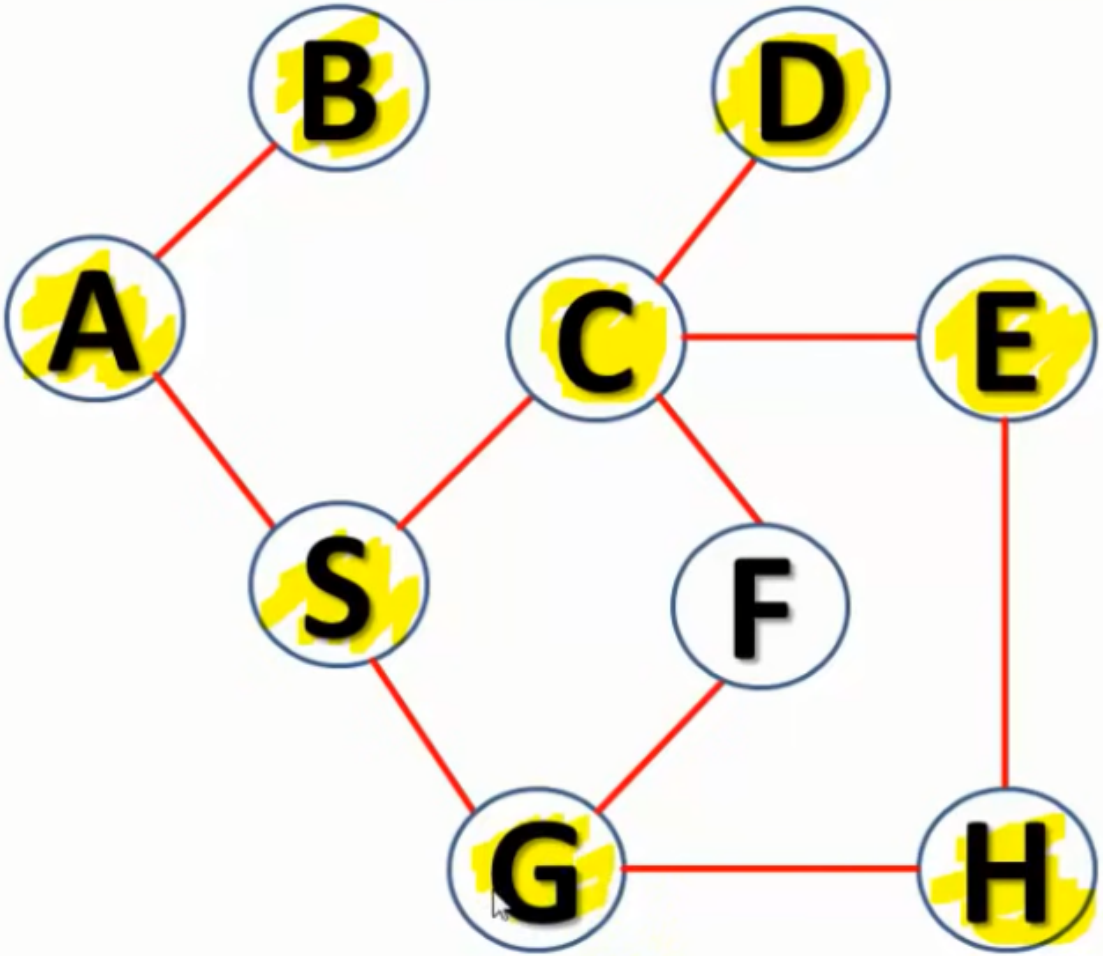
DEPTH FIRST SEARCH



Stack Status

OUTPUT : A B S C D E H

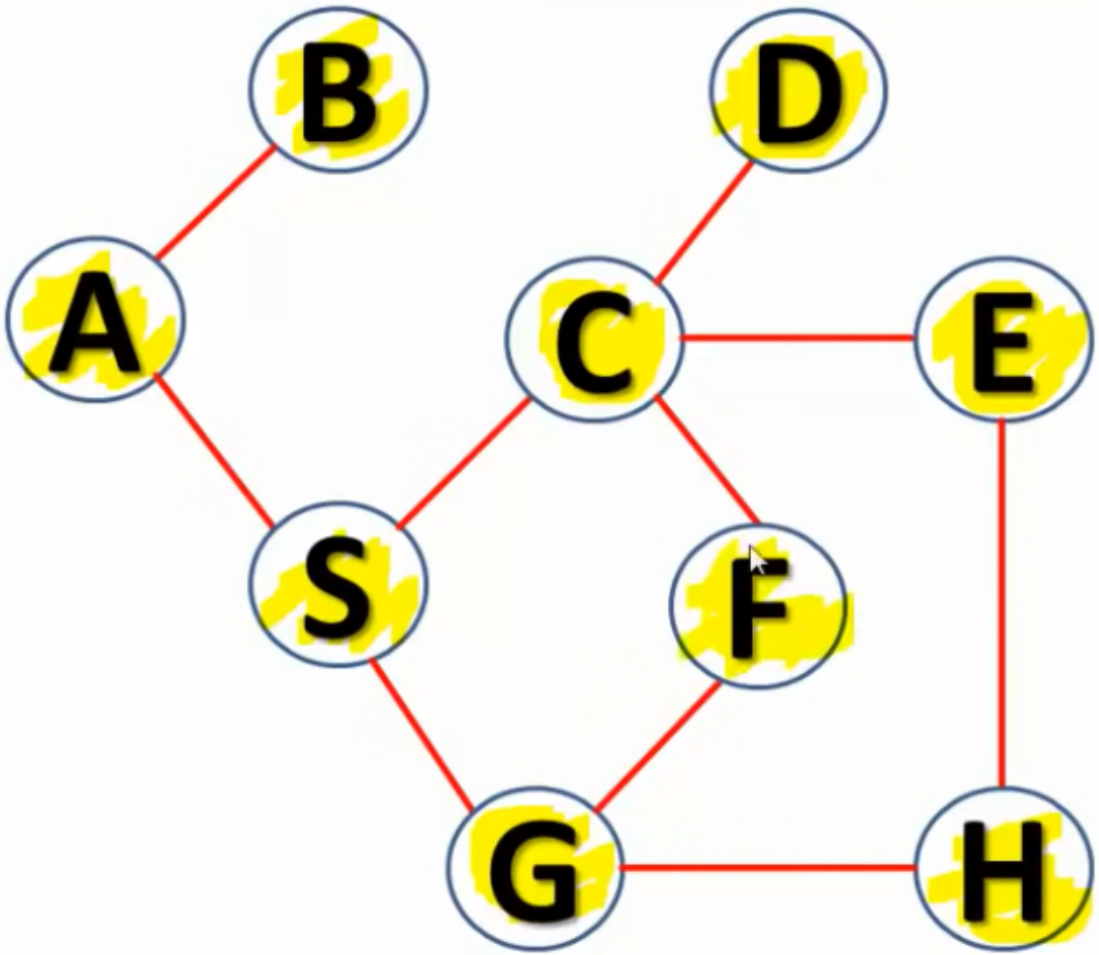
DEPTH FIRST SEARCH



Stack Status

OUTPUT : A B S C D E H G

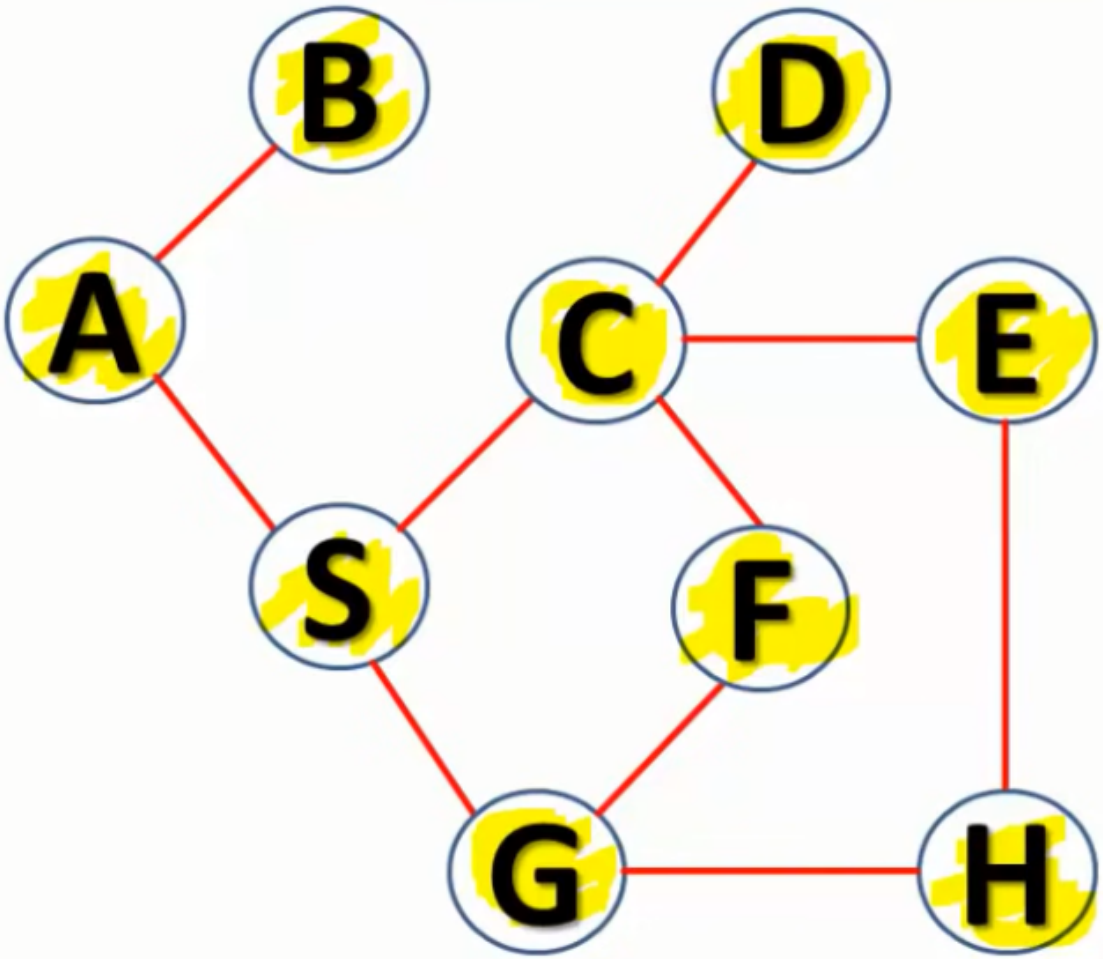
DEPTH FIRST SEARCH



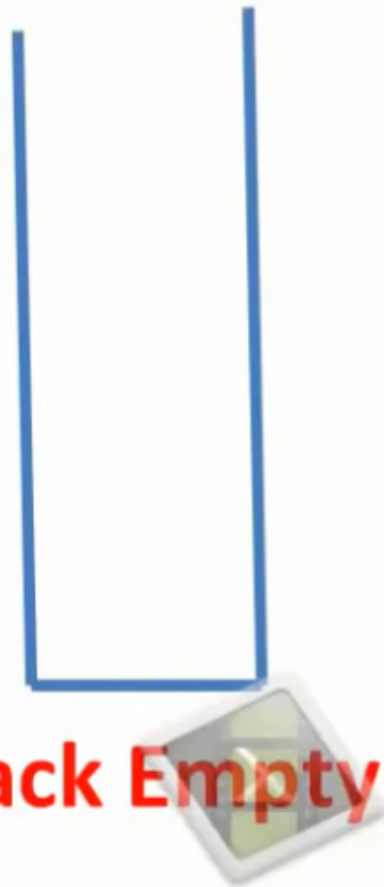
Stack Status

OUTPUT : A B S C D E H G F

DEPTH FIRST SEARCH



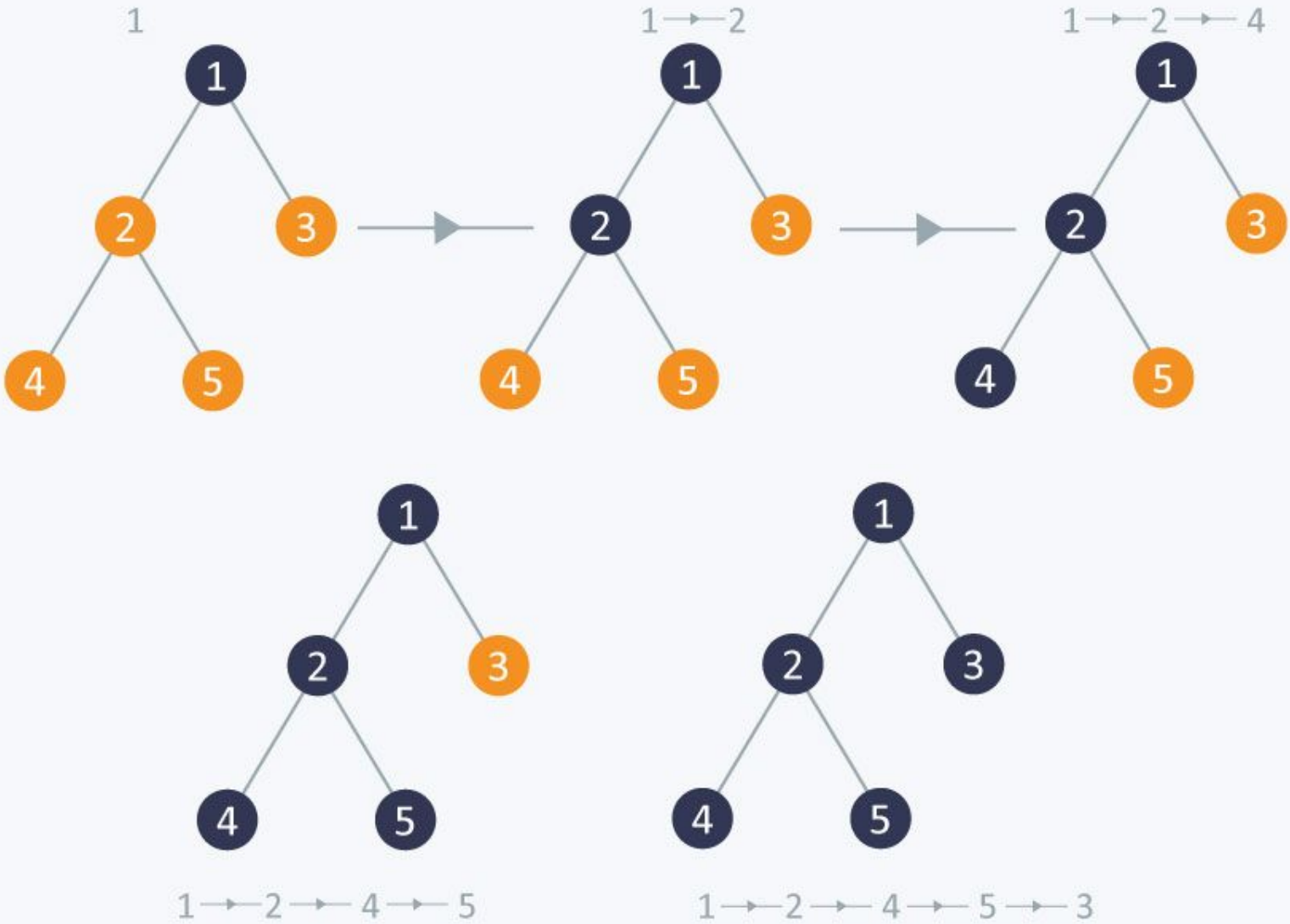
Stack Status



Stack Empty

OUTPUT : A B S C D E H G F

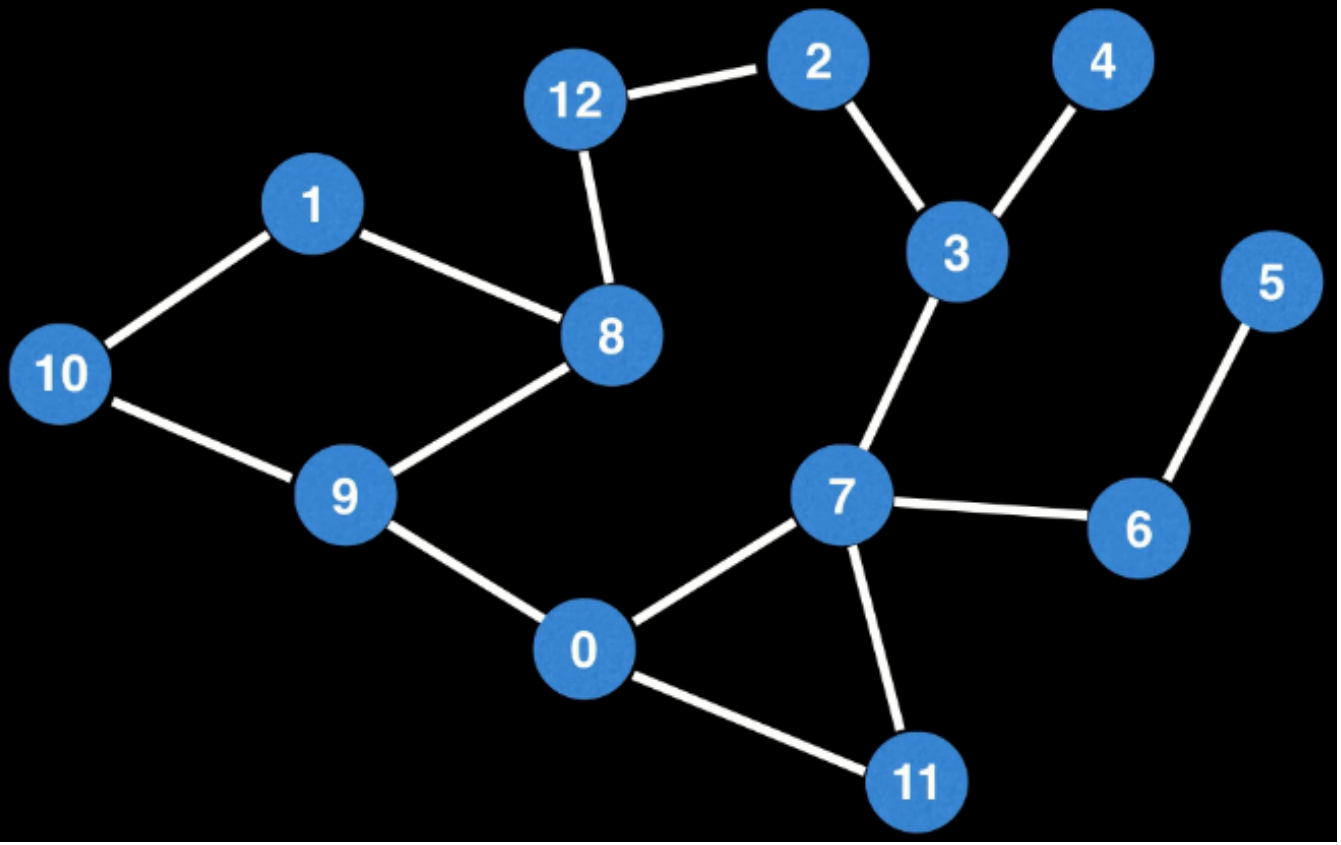
DFS



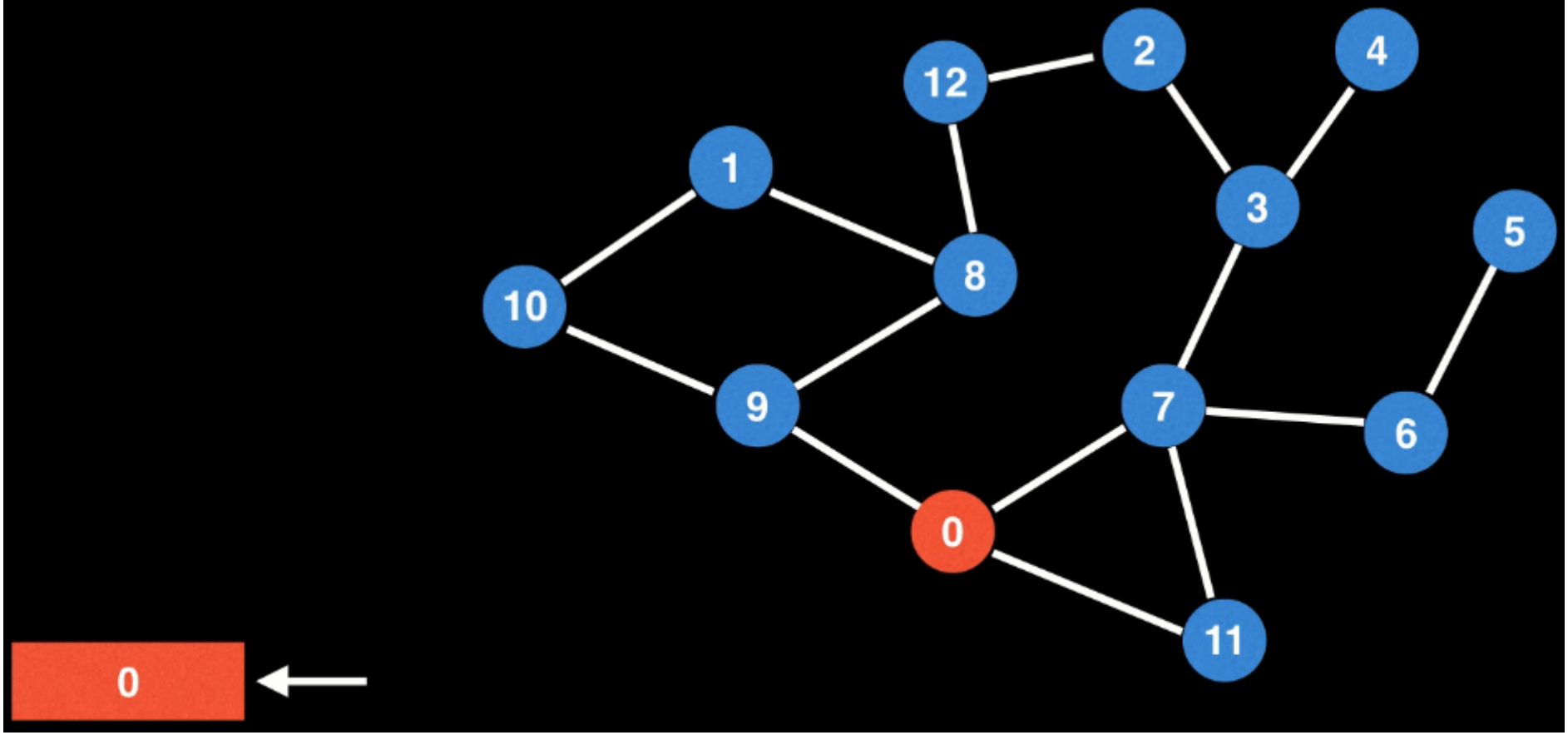
Breadth-First Search (BFS)

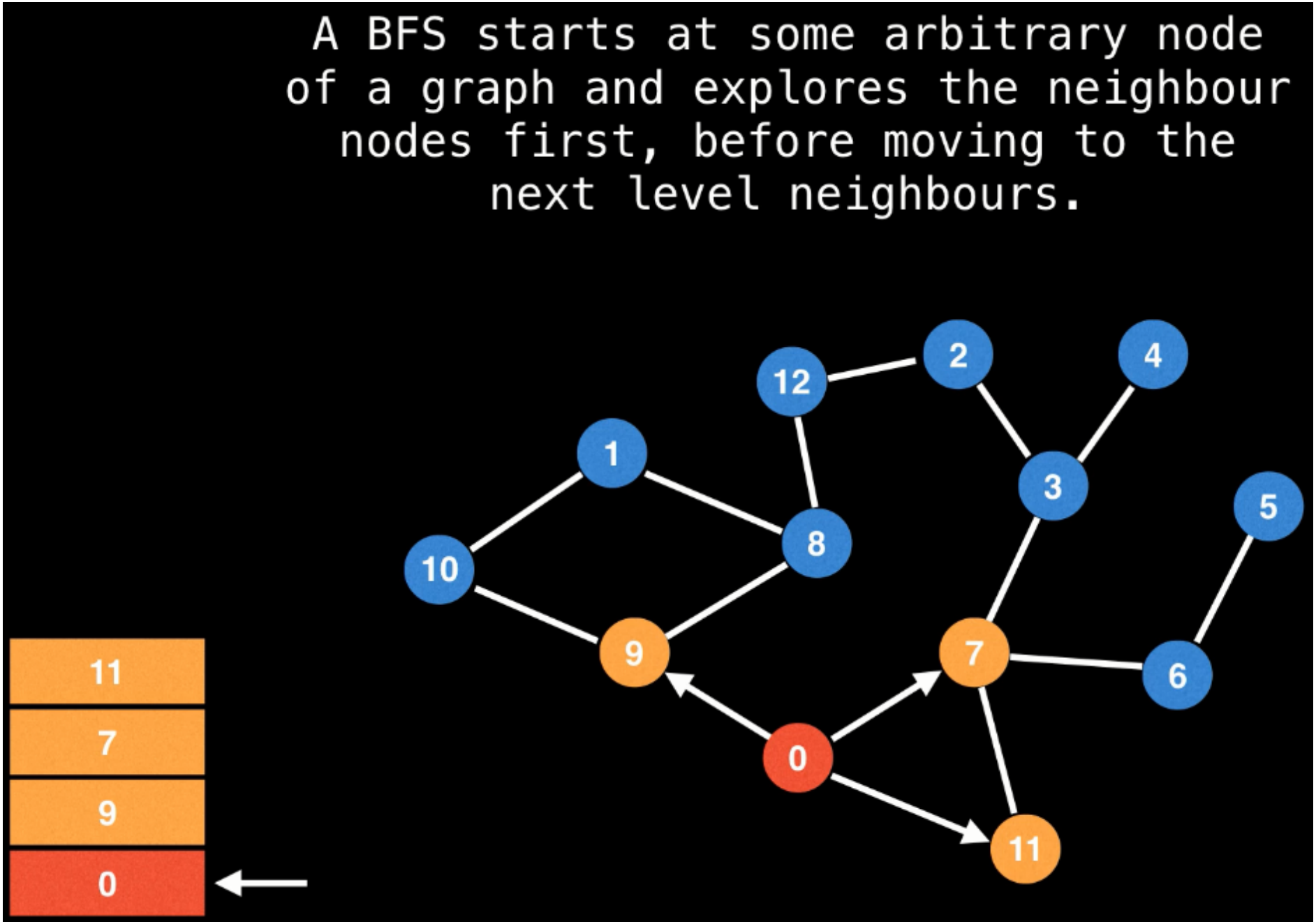
- Start at a given node.
- Mark the starting node as visited and enqueue it.
- Explore all neighbors of the current node.
- Continue until all reachable nodes are visited.

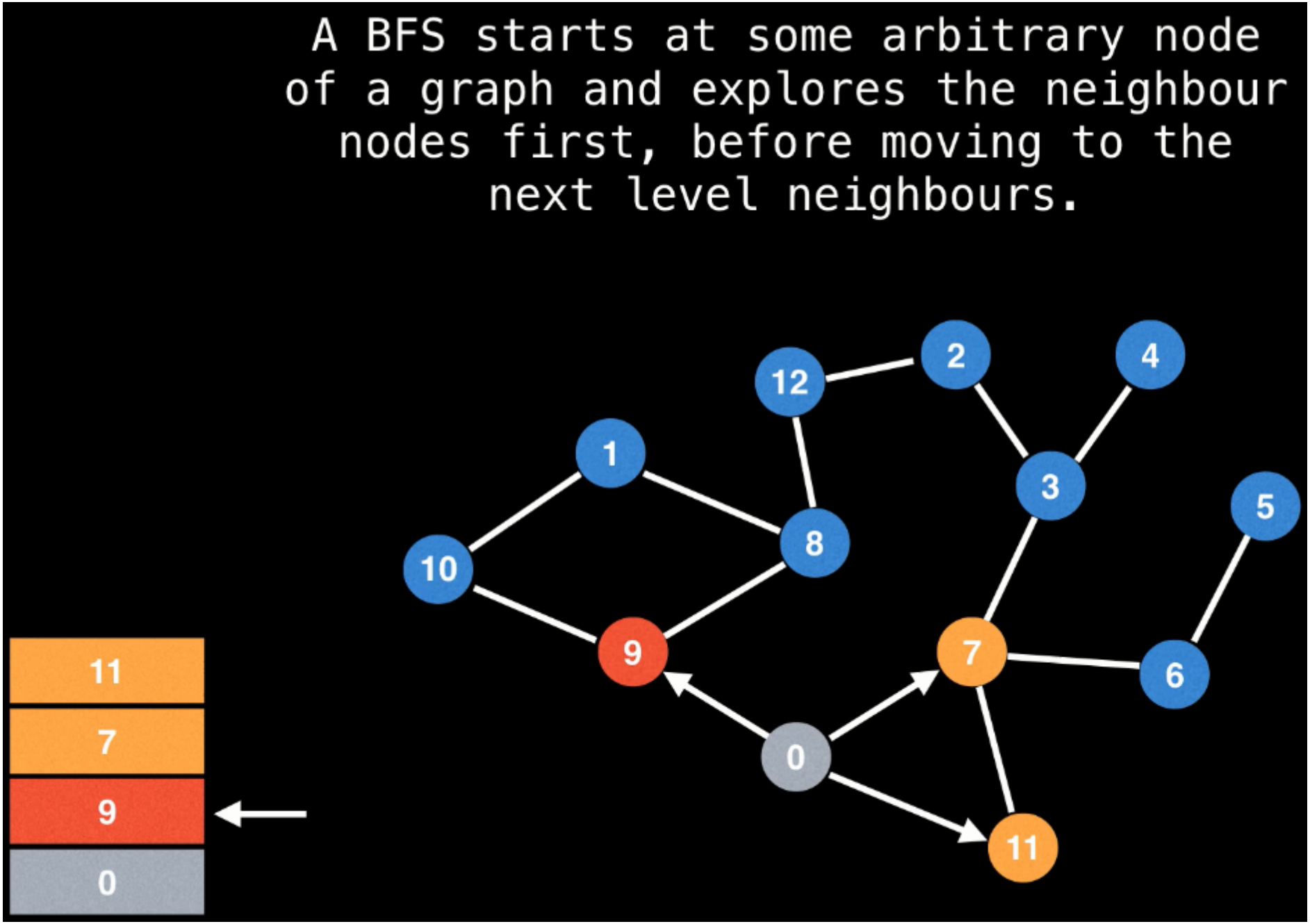
A BFS starts at some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.

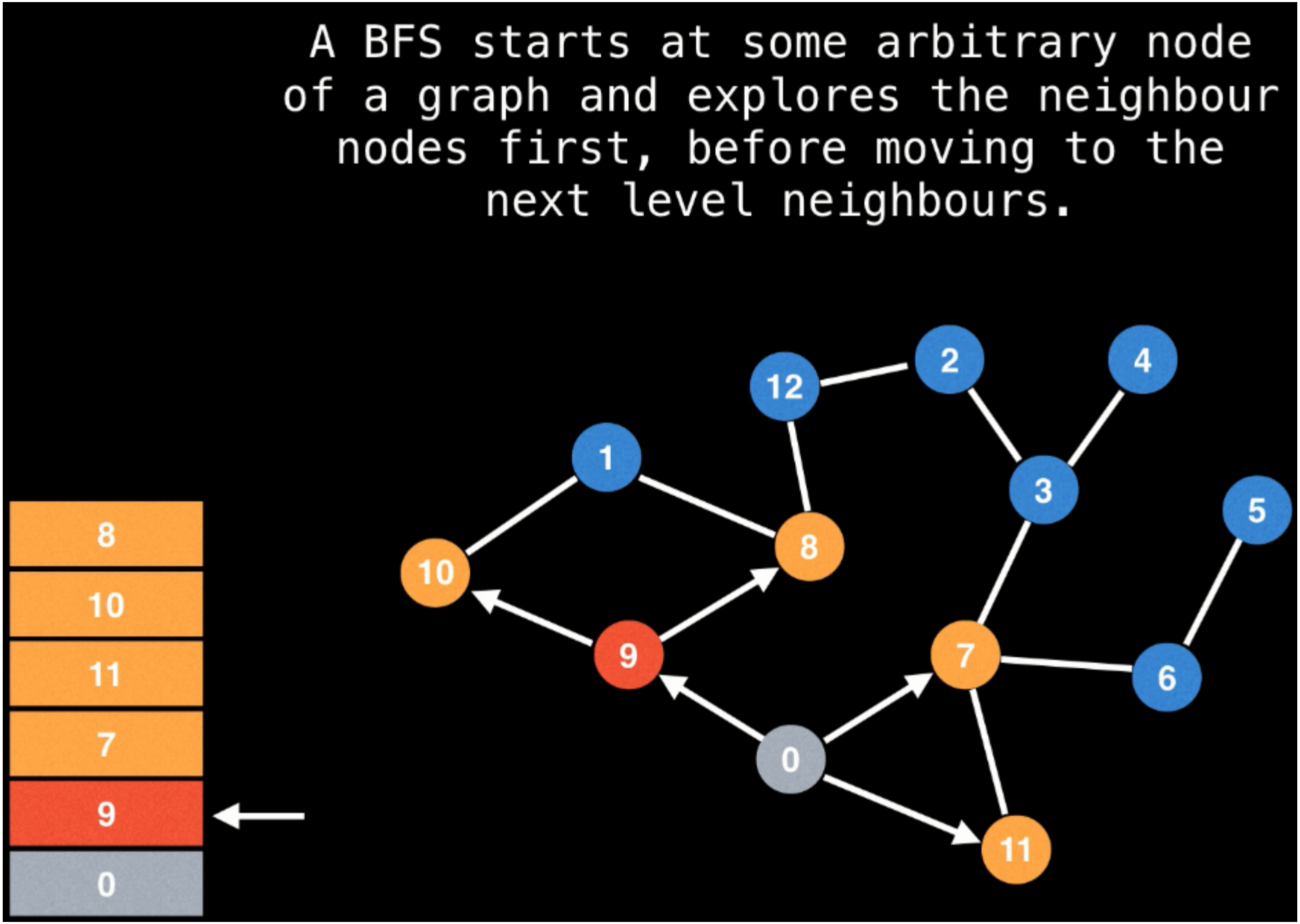


A BFS starts at some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.

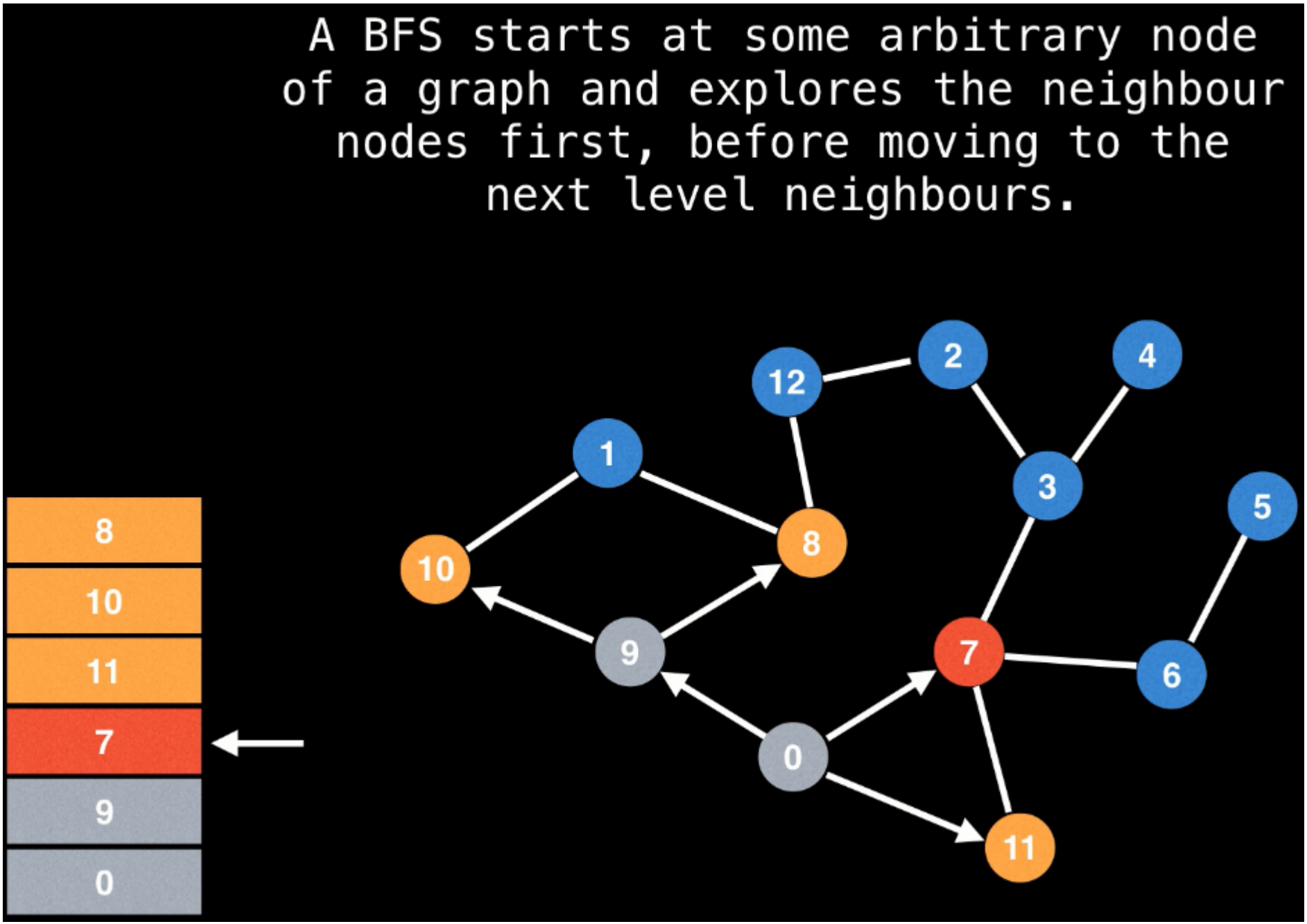


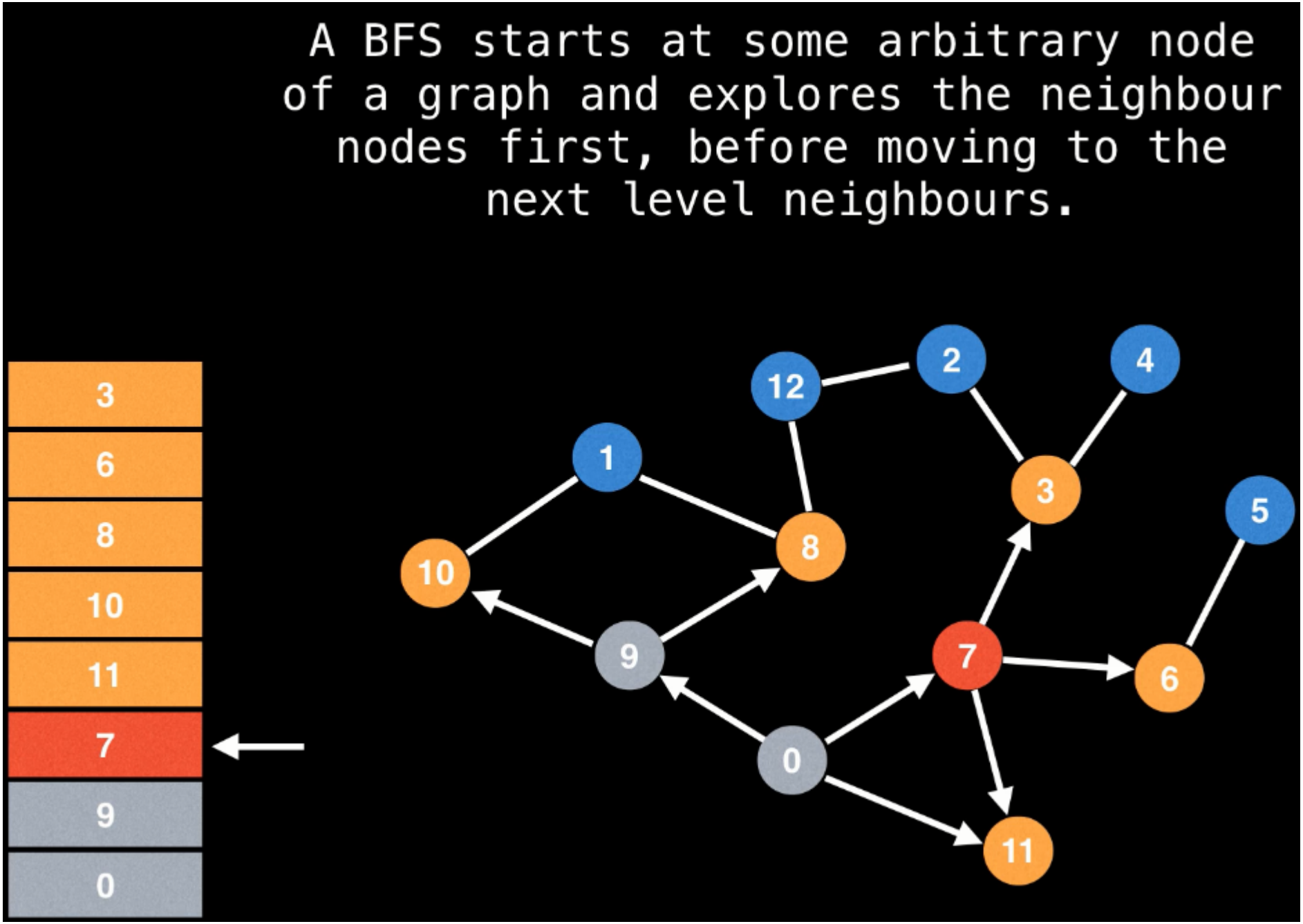

A BFS starts at some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.
12
2
4
1
3
5
8
10
9
7
6
0
11
11
7
9
0

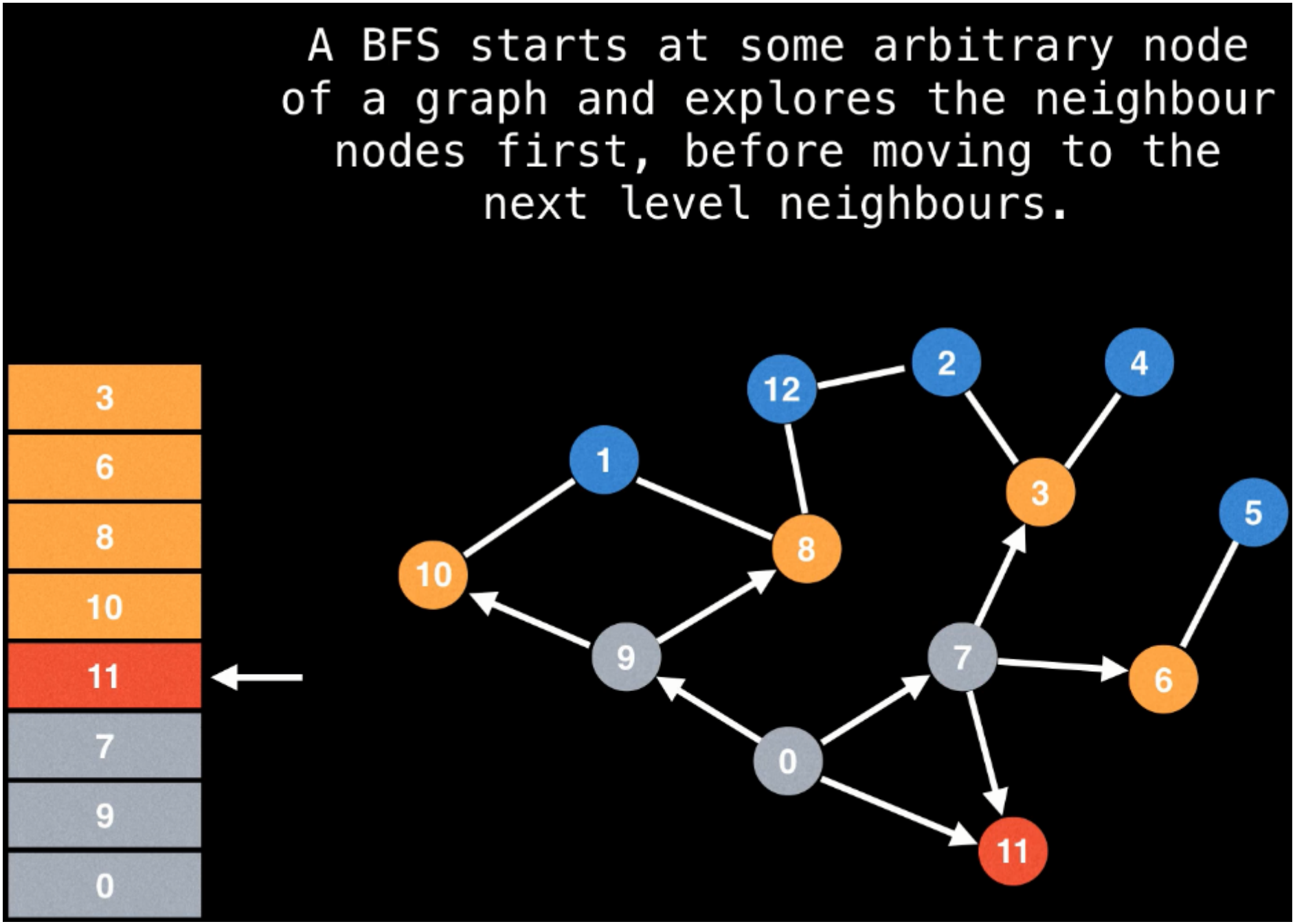

A BFS starts at some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.
12
2
4
1
3
5
8
10
9
7
6
0
11
11
7
9
0


A BFS starts at some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.
8
10
11
7
9
0
12
2
4
1
3
5
8
10
9
7
6
0
11

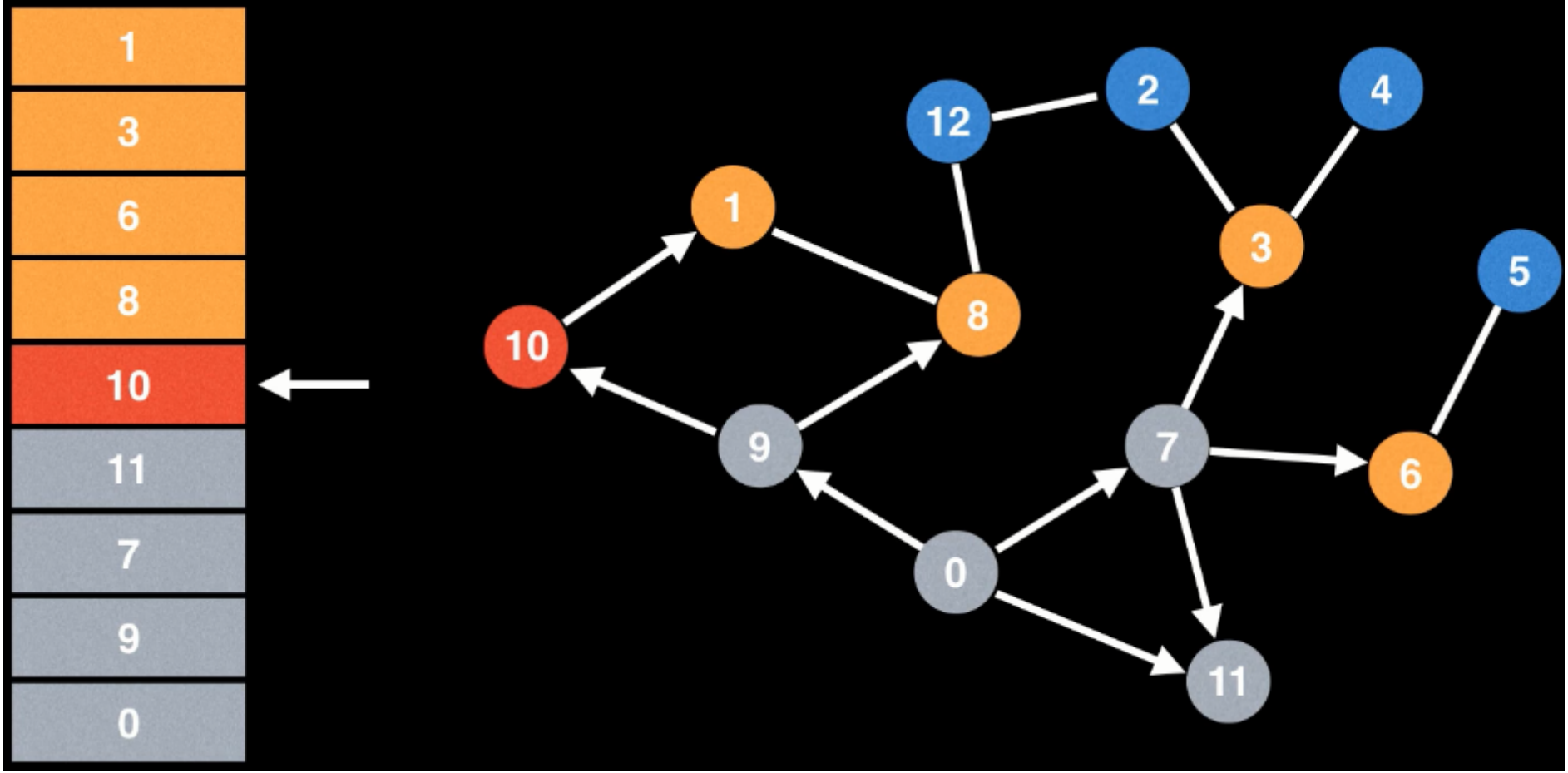
A BFS starts at some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.



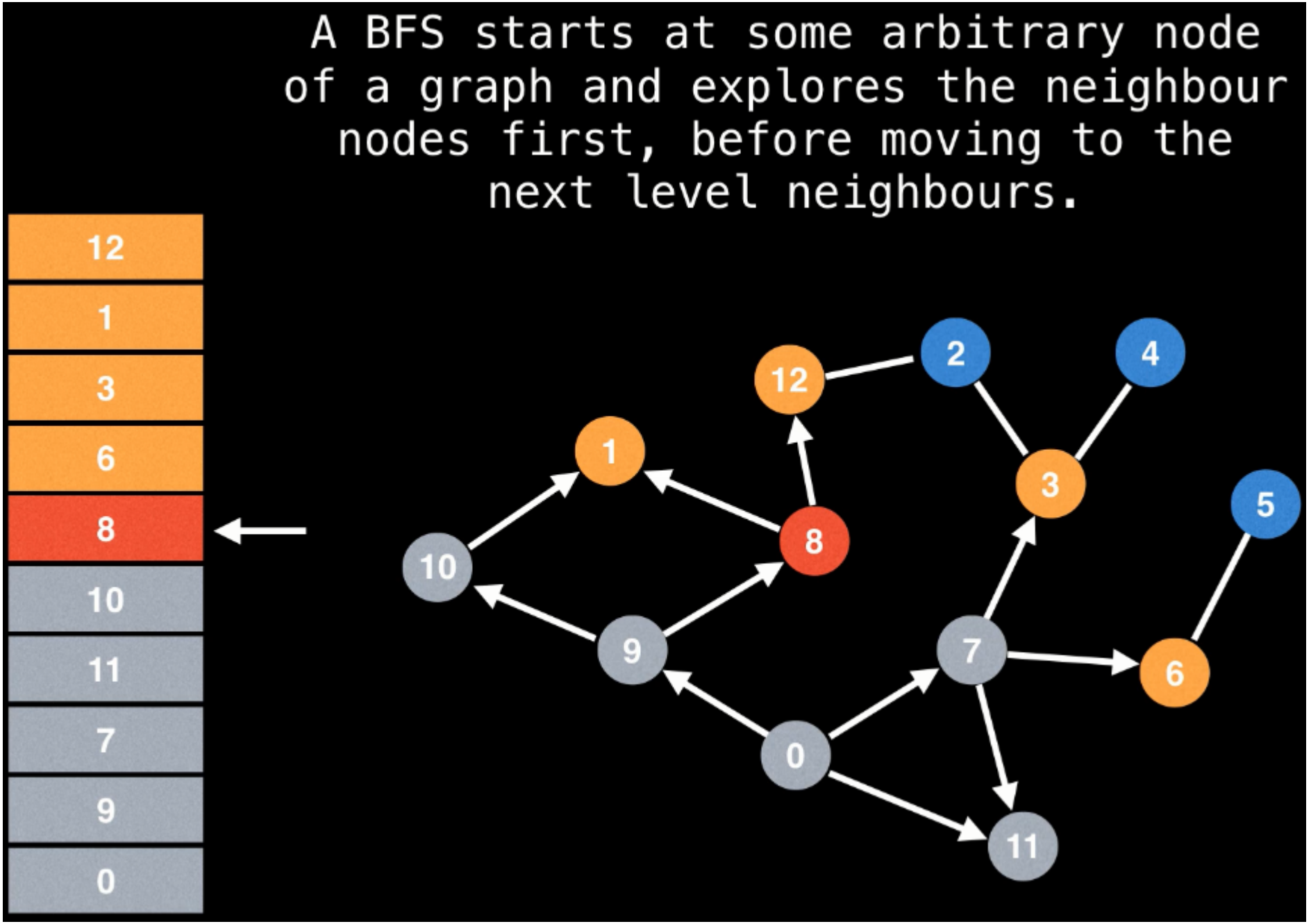

A BFS starts at some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.
3
6
8
10
11
7
9
0
12
2
4
1
3
5
10
8
9
7
6
0
11

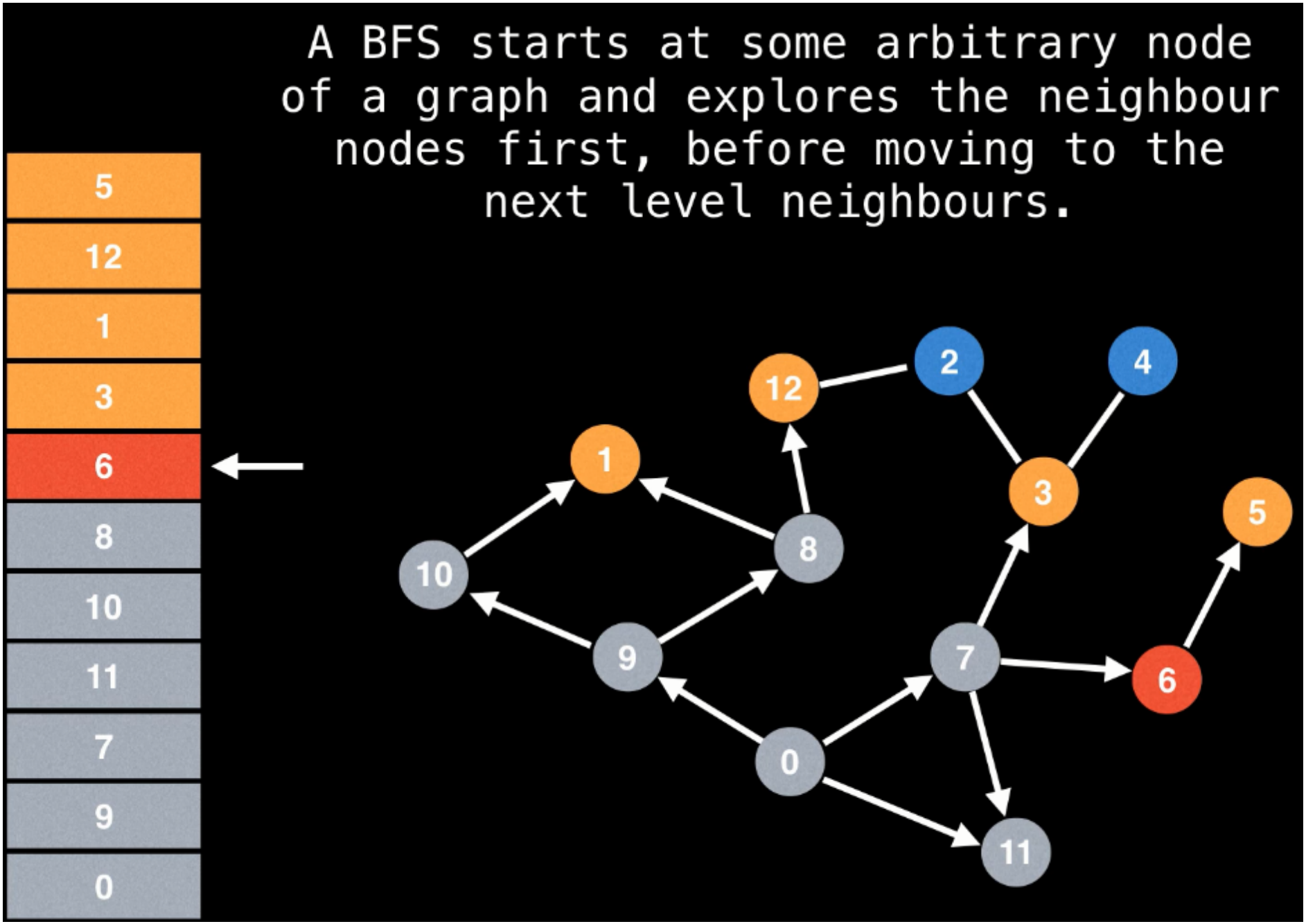

A BFS starts at some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.
3
6
8
10
11
7
9
0
12
2
4
1
3
5
10
8
9
7
6
0
11

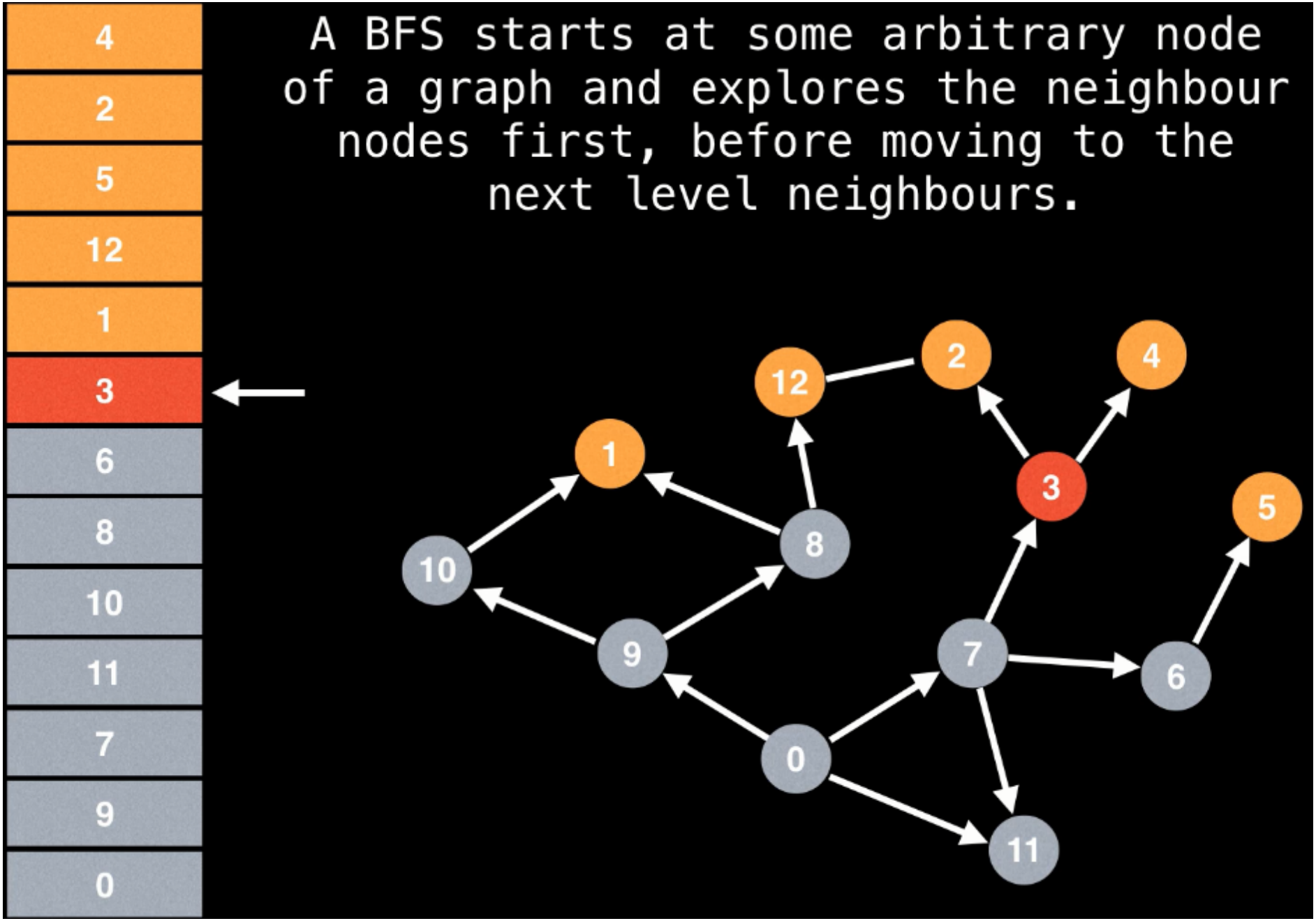
A BFS starts at some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.

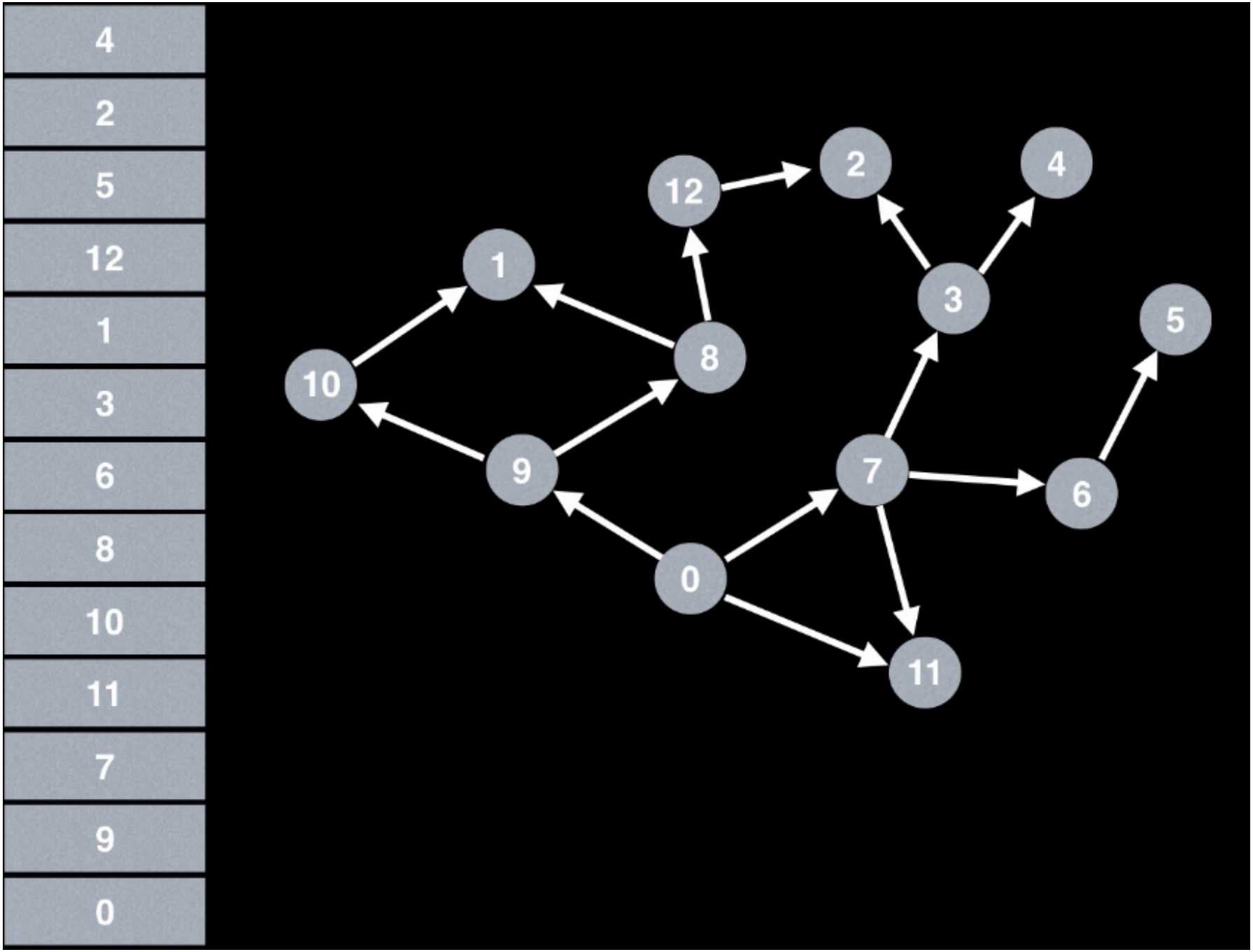


A BFS starts at some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.




A BFS starts at some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.
5
12
1
3
6
8
10
11
7
9
0
2
4
12
1
3
5
8
10
9
7
6
0
11


4
2
5
12
1
3
6
8
10
11
7
9
0
A BFS starts at some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.
12
2
4
1
3
5
10
8
9
7
6
0
11


4
2
5
12
1
3
6
8
10
9
0
7
11
12
2
4
1
3
5
8
10
9
7
6
0
11

BFSV Video Link

Play Breath First Search Video

Depth-First Search (DFS)

1. Start at a given node.
2. Mark the current node as visited.
3. Explore each unvisited neighbor recursively before backtracking.
4. Continue until all reachable nodes are visited.

We only visit nodes that are **reachable** from **already visited** nodes.

A vertex v is considered visited if and only if there exists a vertex u such that:

1. There is an edge from u to v.
2. u has already been visited
3. v itself has not been visited yet.

A vertex v is considered visited if and only if there exists a vertex u such that:

1. There is an edge from u to v.
2. u has already been visited
3. v itself has not been visited yet.

$v \in Visited$?

1. $\exists u | Edge(u, v)$
2. $Visited(u)$
3. $\sim Visited(v)$

$$\forall\, v(Visited(v)) \iff \exists u(Edge(u, v) \wedge Visited(u) \wedge \sim Visited(v))$$

A vertex is marked as visited

A vertex is only marked as visited if it can be reached from another visited vertex or is already a visited node itself.

$$\forall u, v\, (Visited(u) \rightarrow Visited(u) \vee Edge(u, v))$$

DFS in First-Order Logic

Predicates:

- Vertex(v): v is a vertex in the graph.
- Edge(u, v): There is an edge from u to v.
- Visited(v): Vertex v has been visited.
- Push(s, v): Vertex v is pushed onto stack s.
- Pop(s, v): Vertex v is popped from stack s.

Axioms:

$$\forall v(\text{Visited}(v) \leftrightarrow \exists u(\text{Edge}(u, v) \land \text{Visited}(u) \land \neg\text{Visited}(v)))$$

$$\forall u, v(\text{Visited}(v) \rightarrow \text{Visited}(u) \lor \text{Edge}(u, v))$$

Rules:

- If $\neg\text{Visited}(v) \land \text{Edge}(u, v)$, then Push($s, v$).
- If Push(s, v), then Visited(v).

1. If a vertex v is not visited and there is an edge from vertex u to v, then push v onto the stack.

$$\sim Visited(v) \wedge Edge(u, v) \rightarrow Push(s, v).$$

2. If a vertex v is pushed onto the stack, then it should be marked as visited.

$$Push(s, v) \rightarrow Visited(v)$$

Breadth-First Search (BFS)

- Start at a given node.
- Mark the starting node as visited and enqueue it.
- Explore all neighbors of the current node.
- Continue until all reachable nodes are visited.

BFS in First-Order Logic

Predicates:

- Vertex(v): v is a vertex in the graph.
- Edge(u, v): There is an edge from u to v.
- Visited(v): Vertex v has been visited.
- Enqueue(q, v): Vertex v is enqueued in queue q.
- Dequeue(q, v): Vertex v is dequeued from queue q.

Axioms:

$$\forall v(\text{Visited}(v) \leftrightarrow \exists u(\text{Edge}(u, v) \land \text{Visited}(u) \land \neg\text{Visited}(v)))$$

$$\forall u, v(\text{Visited}(v) \rightarrow \text{Visited}(u) \lor \text{Edge}(u, v))$$

Rules:

- If $\neg\text{Visited}(v) \land \text{Edge}(u, v)$, then Enqueue($q, v$).
- If Enqueue(q, v), then Visited(v).

Comparison of DFS and BFS

DFS

1. Uses a stack (either implicitly via recursion or explicitly).
2. Explores as deep as possible along each branch before backtracking.
3. Suitable for exploring all paths in a search space.

BFS

1. Uses a queue to explore all neighbors level by level.
2. Explores all nodes at the current depth before moving to the next.
3. Suitable for finding the shortest path in an unweighted graph.

1. Proposition and Statement
2. Compound Proposition
   - Connectives;
   - Logical Equivalence;
   - Tautology and Contradictions.
3. Conditional/Bi-conditional Statements