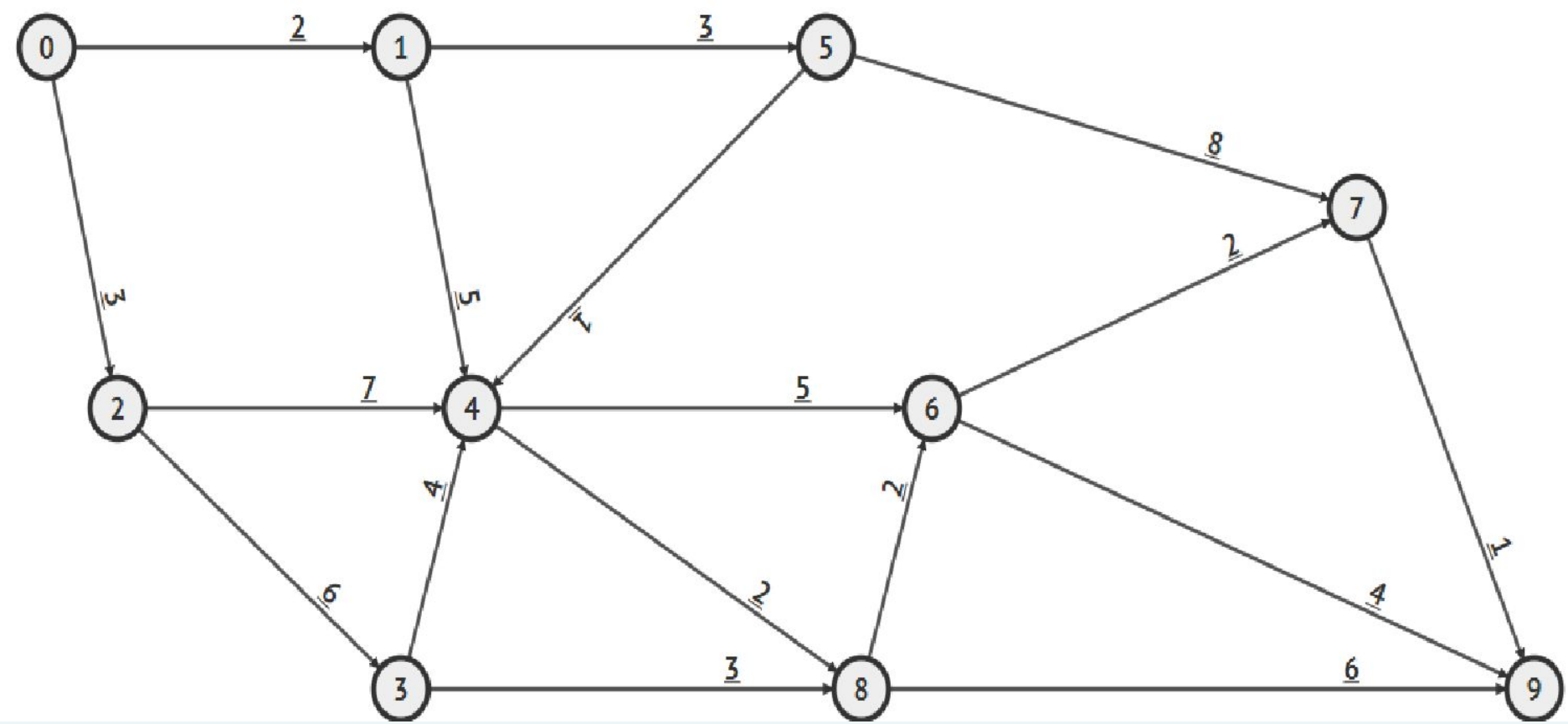
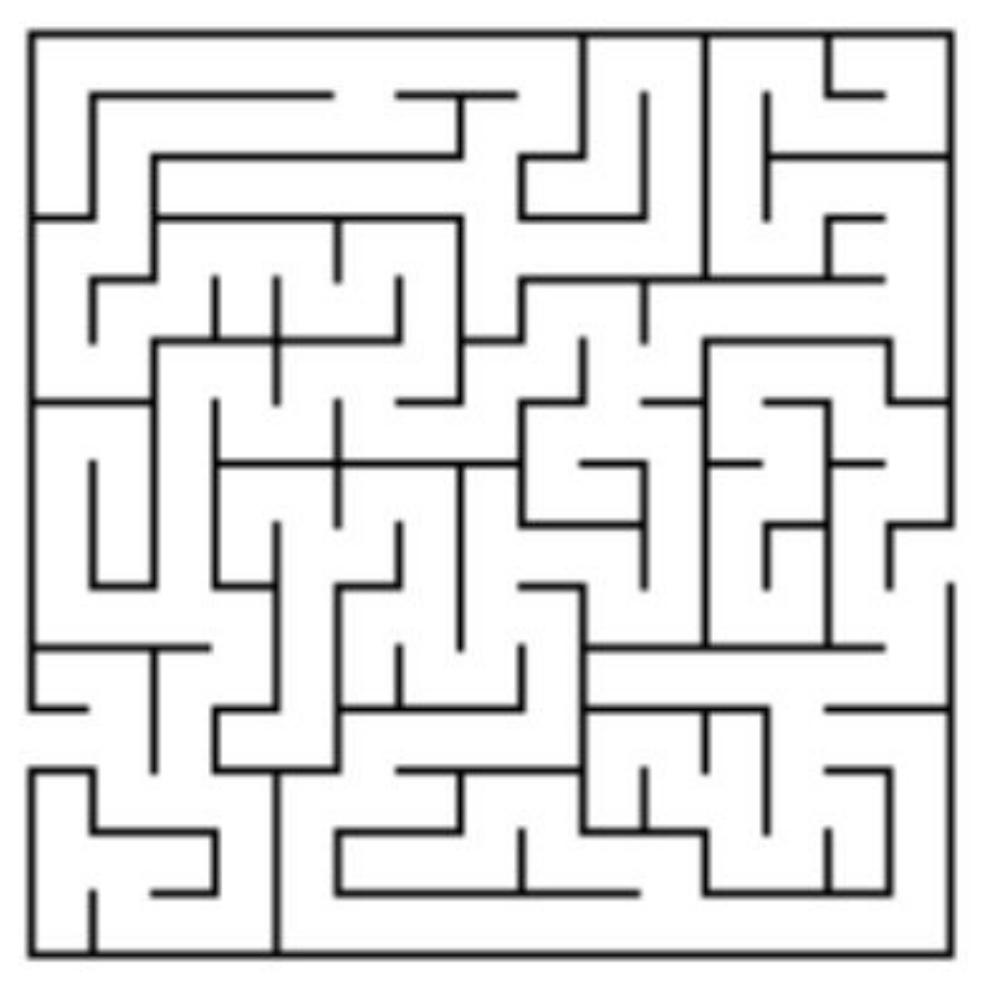
GRAPH TRAVERSAL

Example

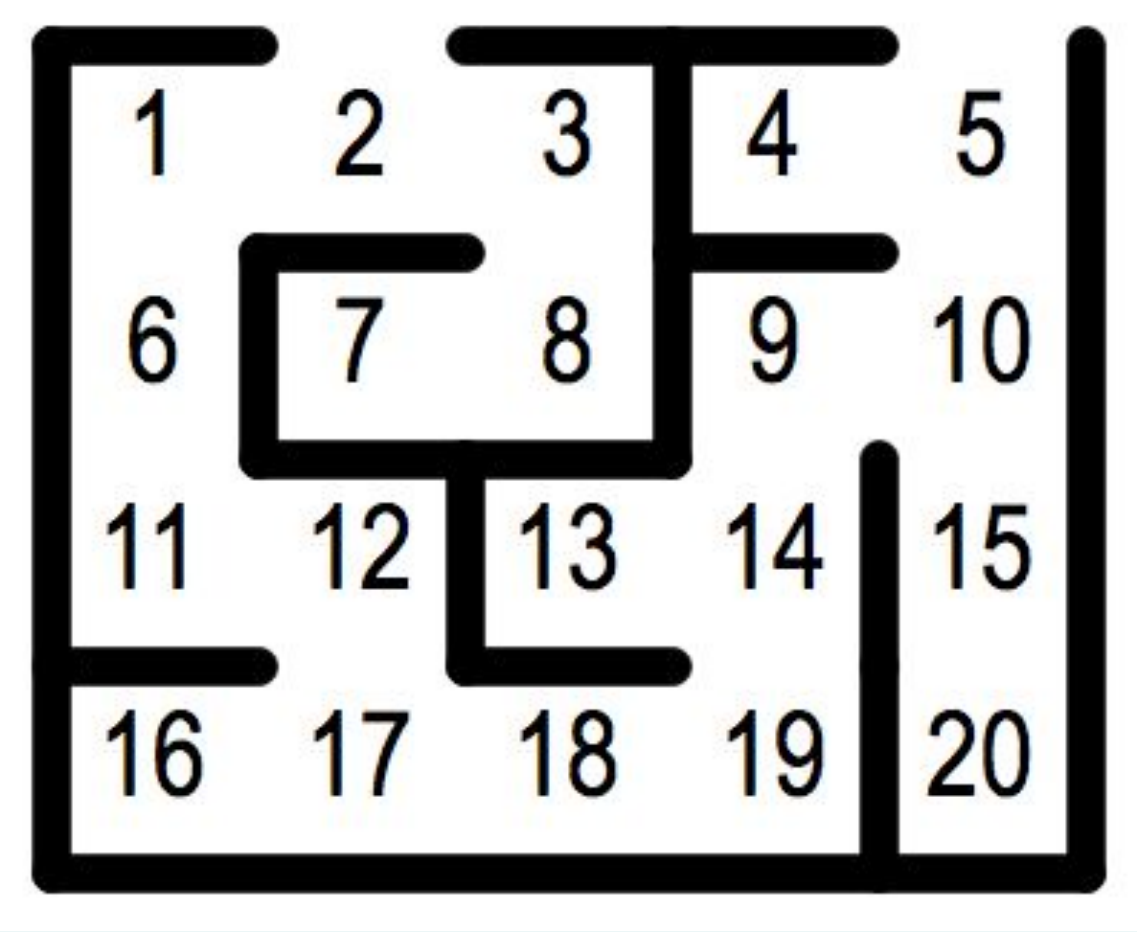
Is there a path from 0 to 9? If there is, list one of them.



Maze Traversal Puzzle



Maze Traversal



Graph Traversal

Graph traversal (also known as graph search) refers to the process of visiting (checking and/or updating) each vertex in a graph. Such traversals are classified by the order in which the vertices are visited

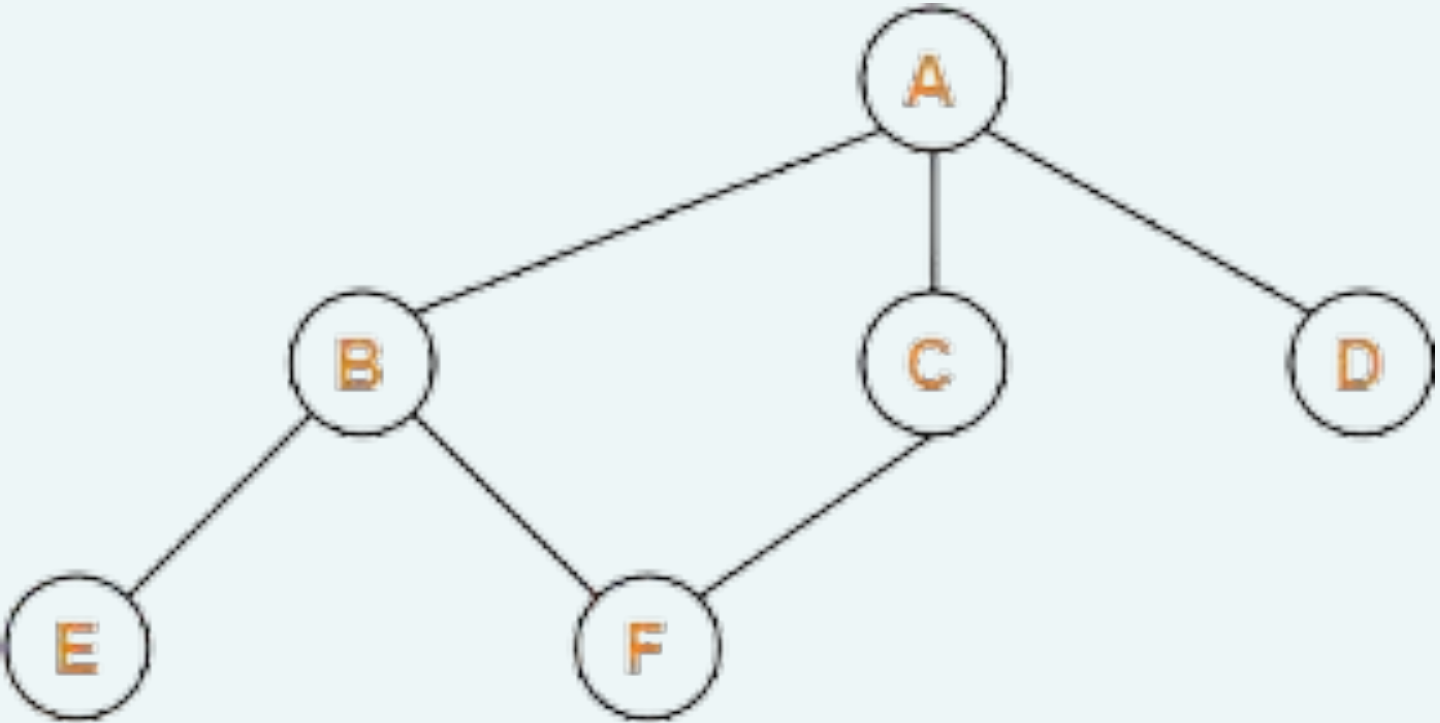

SOCIAL
NETWORK

Depth First Search - DFS

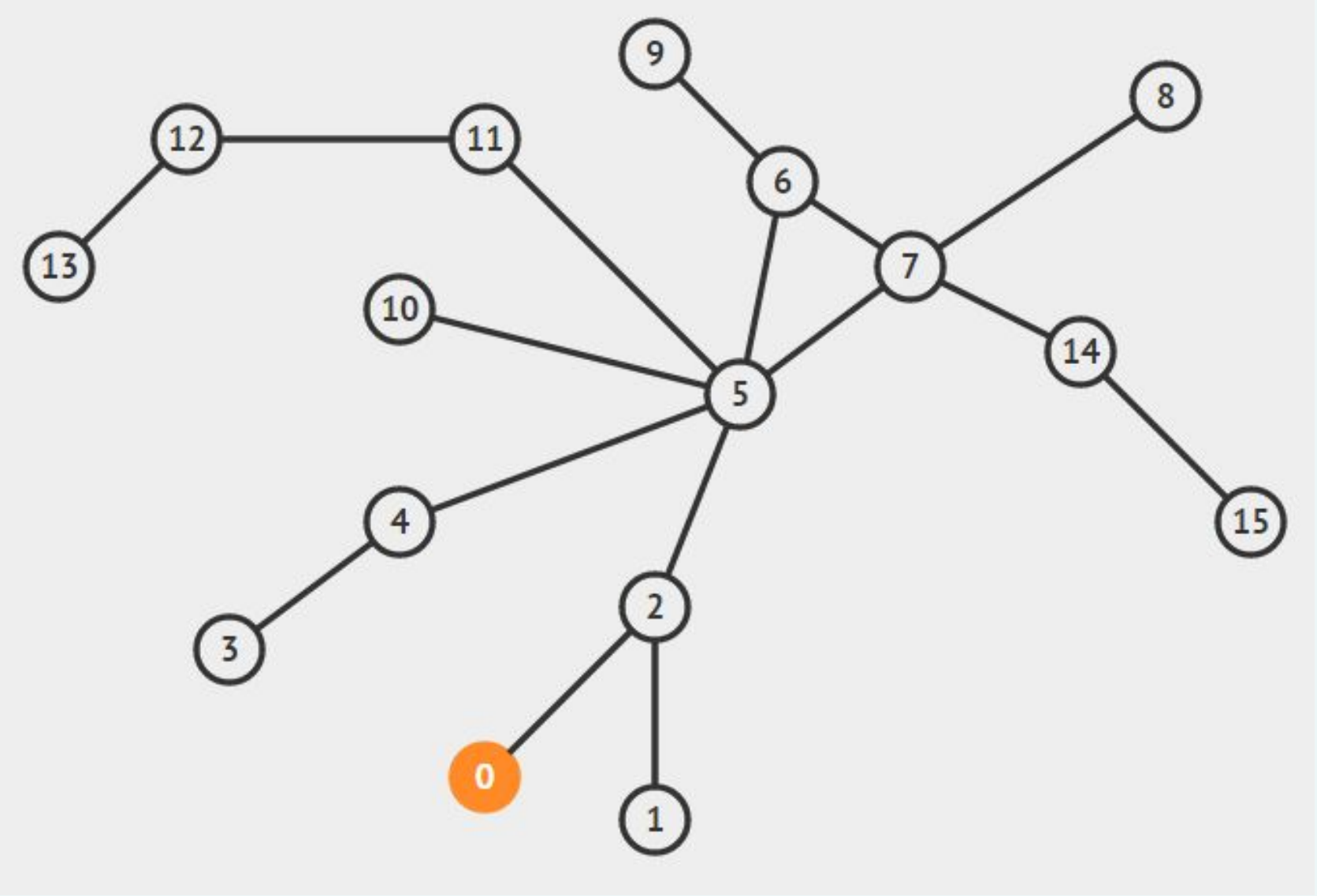
A depth-first search (DFS) is an algorithm for traversing a finite graph. DFS visits the child vertices before visiting the sibling vertices; that is, it traverses the depth of any particular path before exploring its breadth. A stack (often the program's call stack via recursion) is generally used when implementing the algorithm.

```
procedure DFS(G,v):
    label v as discovered
    for all edges from v to w in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G,w)
```

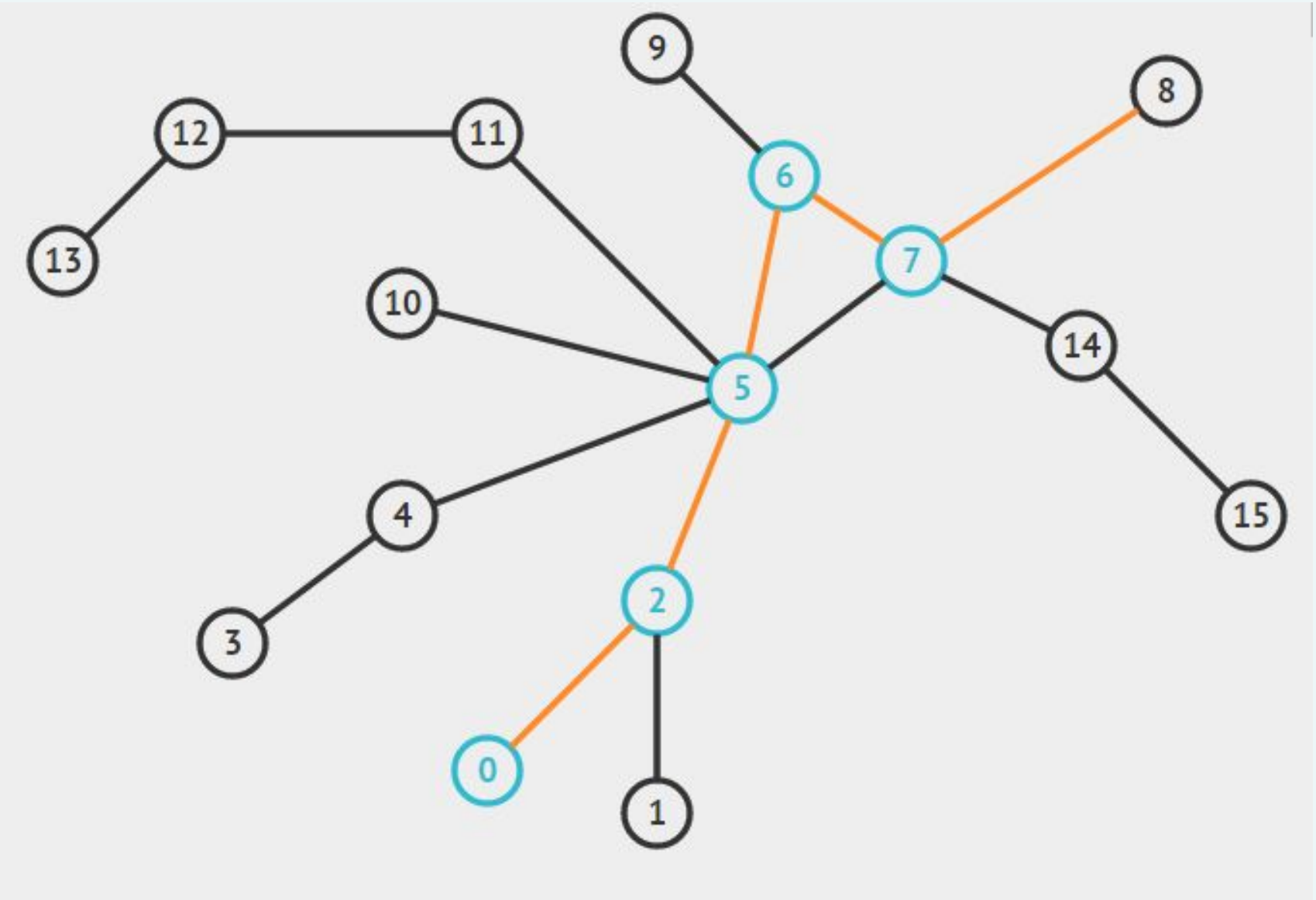
```
procedure DFS(G,v):
    label v as discovered
    for all edges from v to w in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G,w)
```



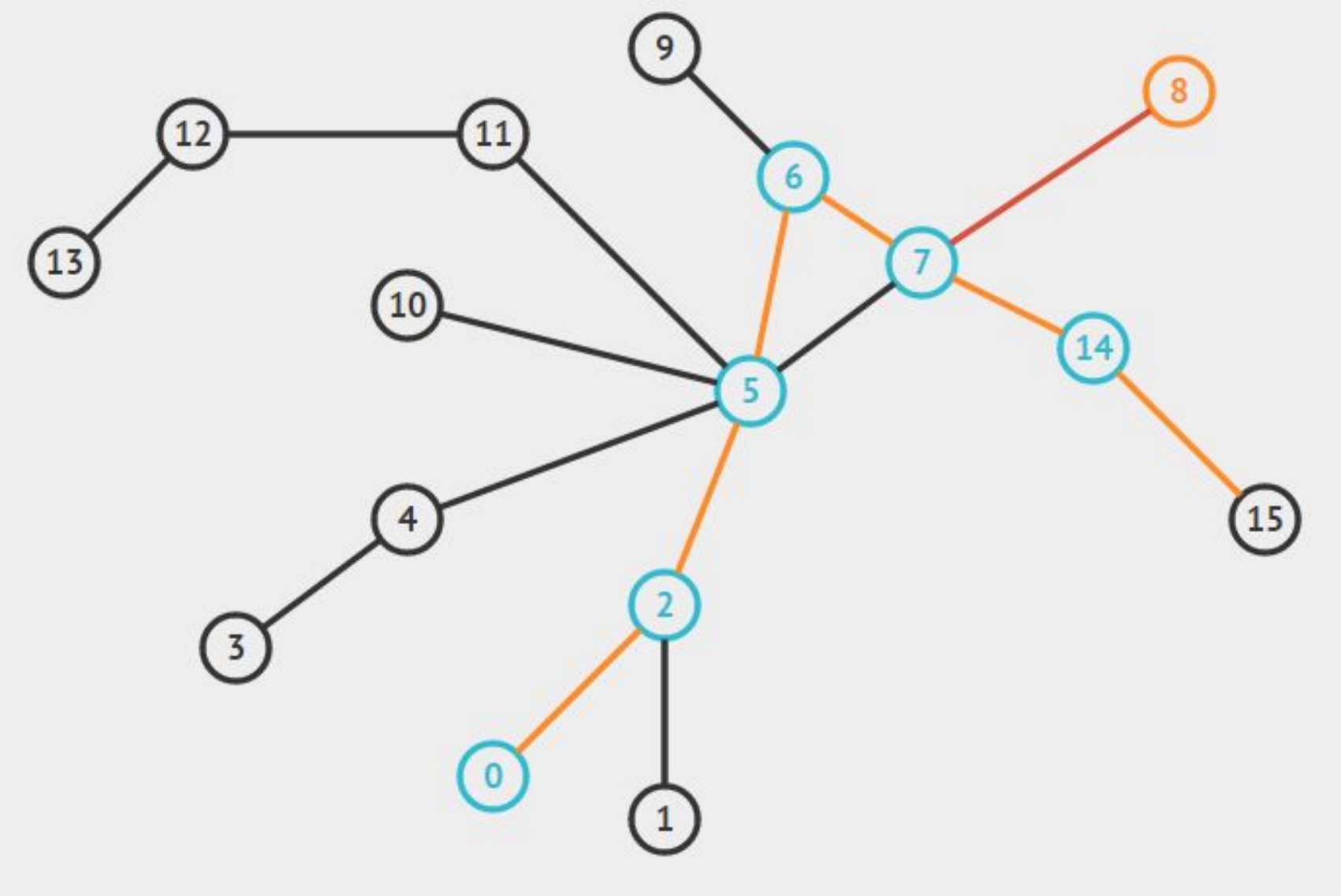
Depth First Search - DFS



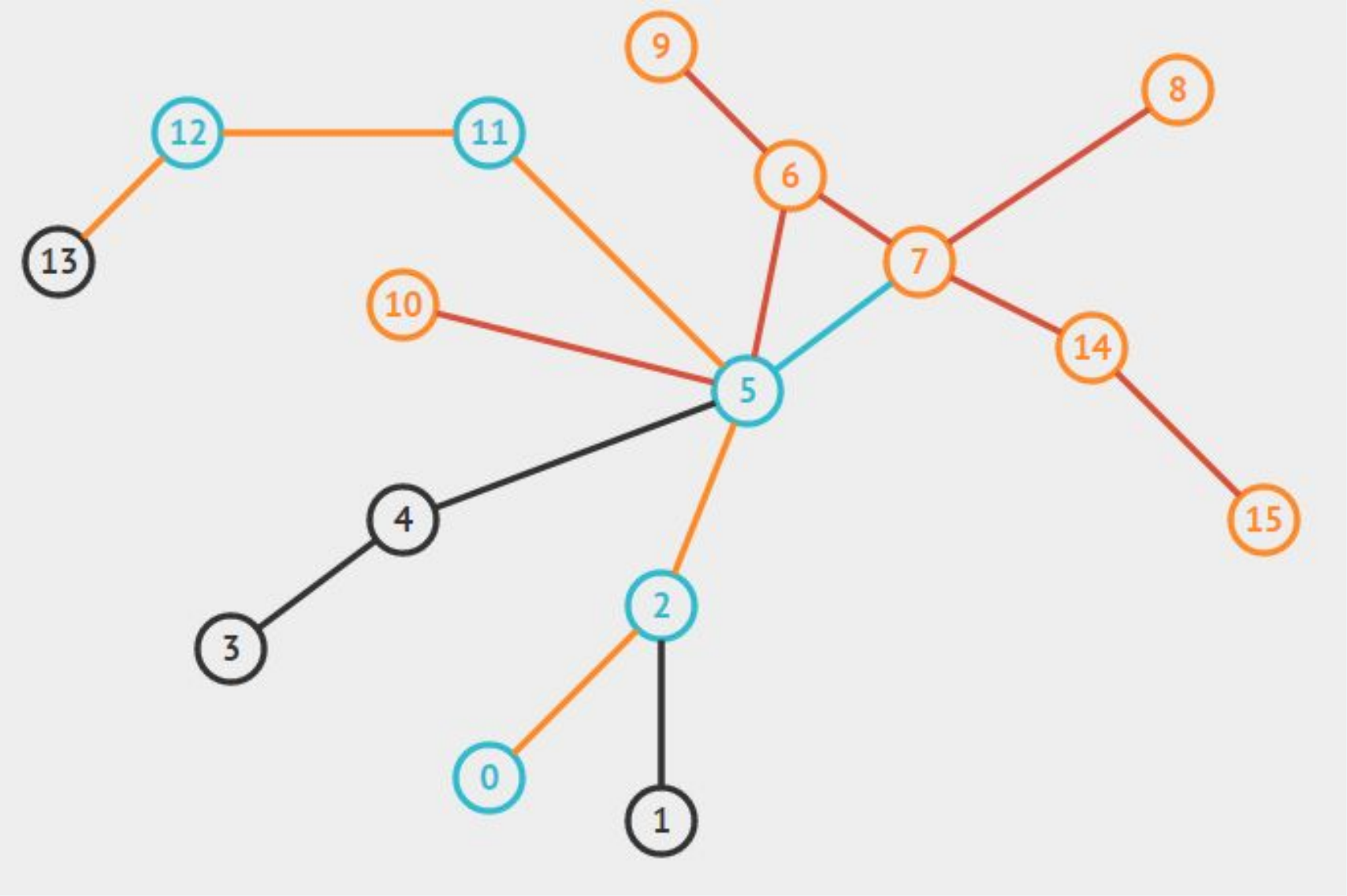
Depth First Search - DFS



Depth First Search - DFS

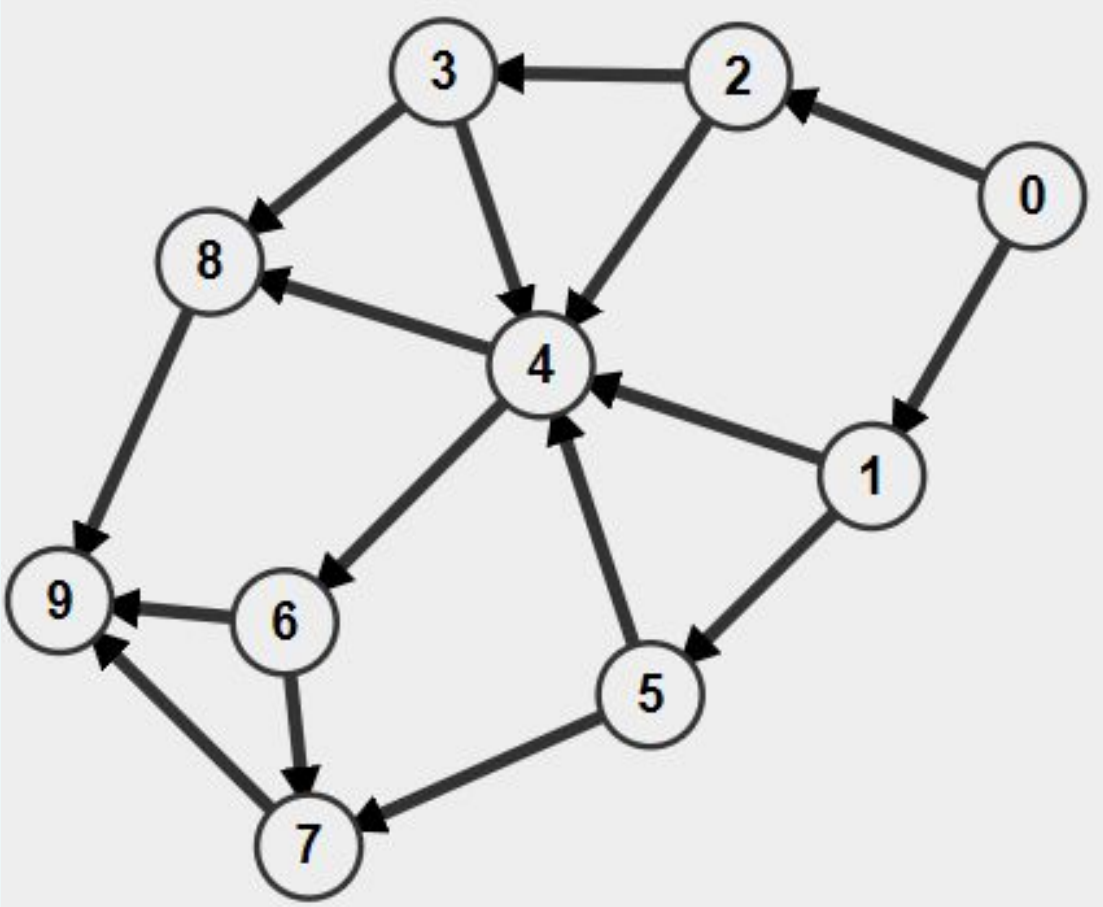


Depth First Search - DFS



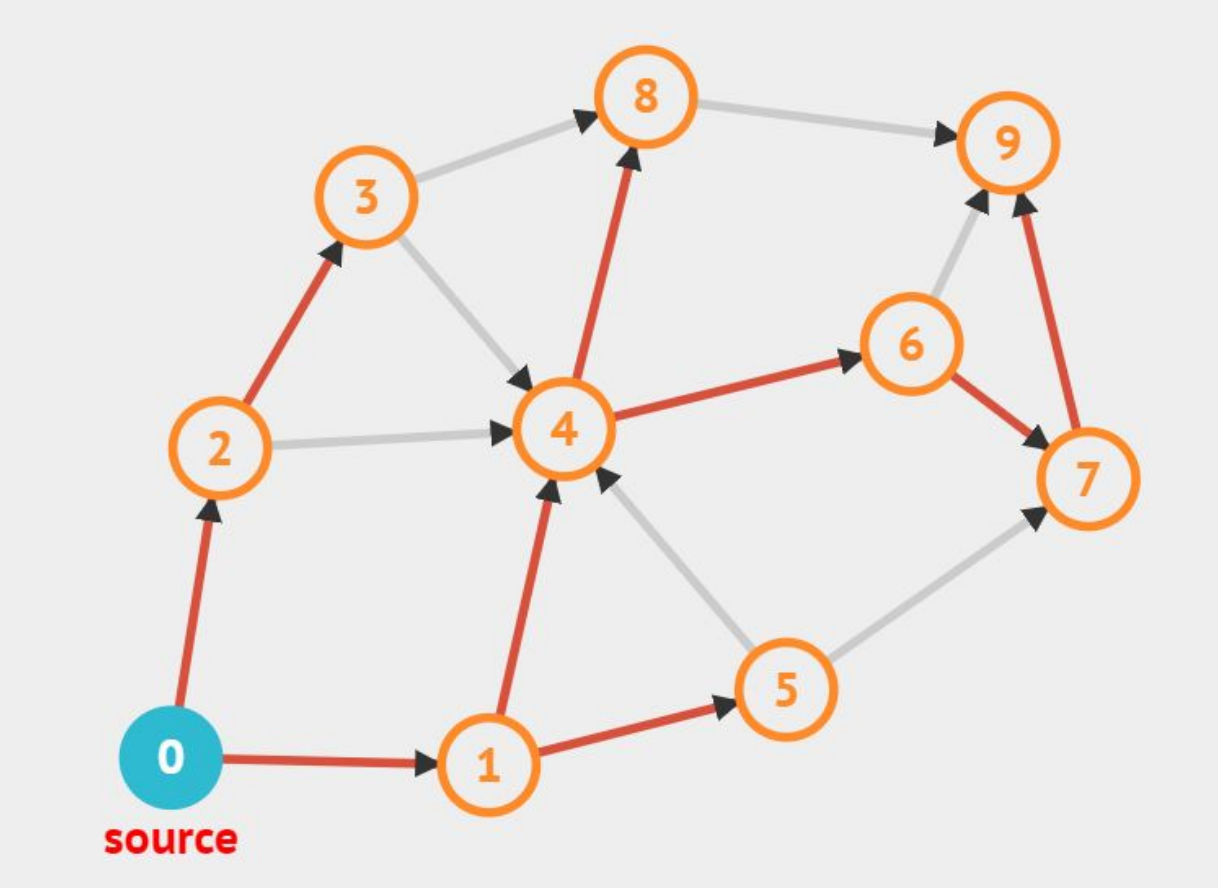
DFS

- Print the lexicographically largest DFS of the following graph starting from 0



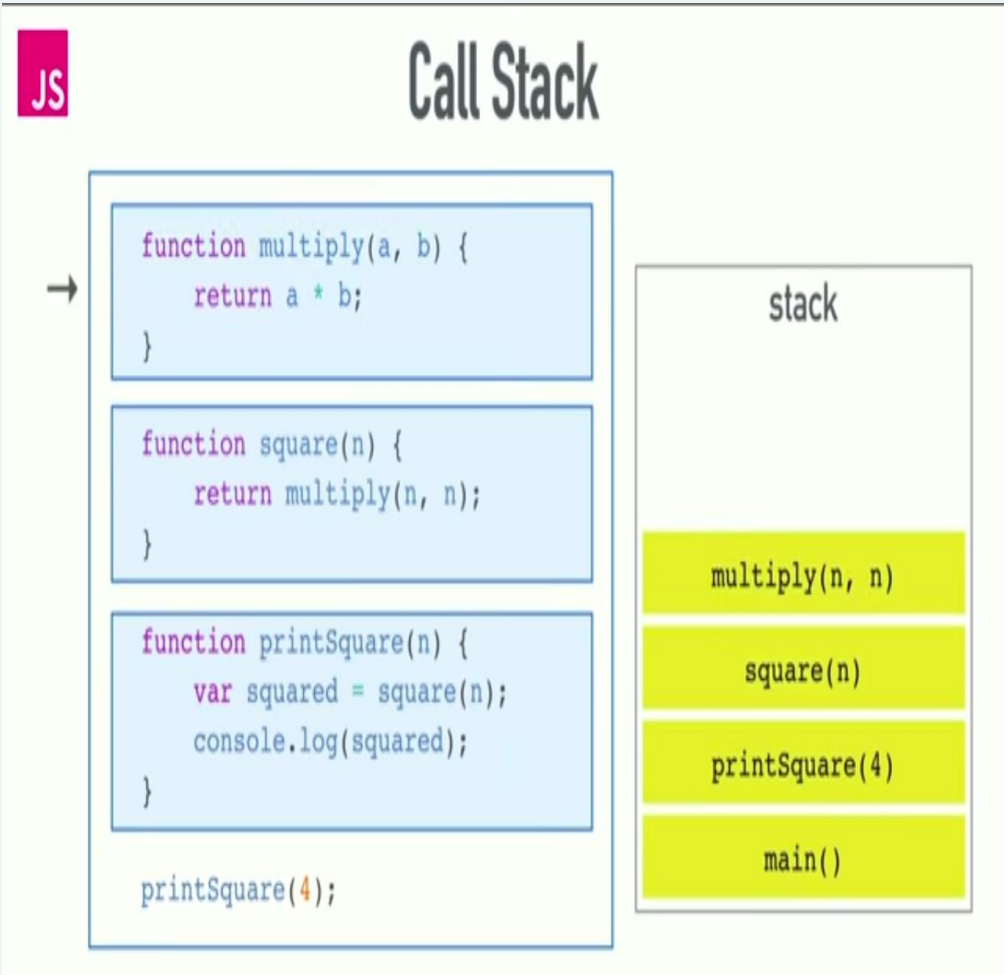
DFS

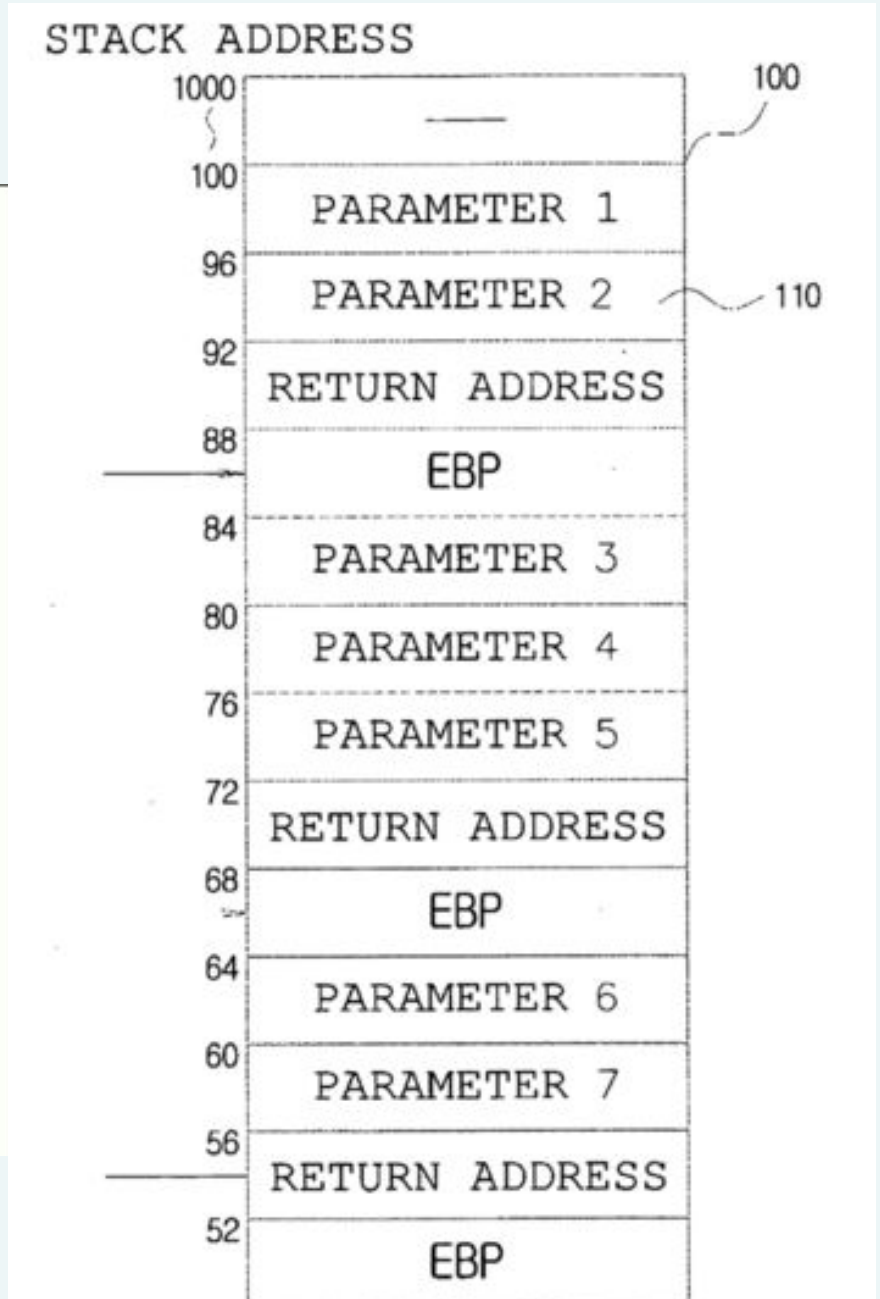
- Print the lexicographically smallest DFS of the following graph



0 1 4 6 7 9 8 5 2 3

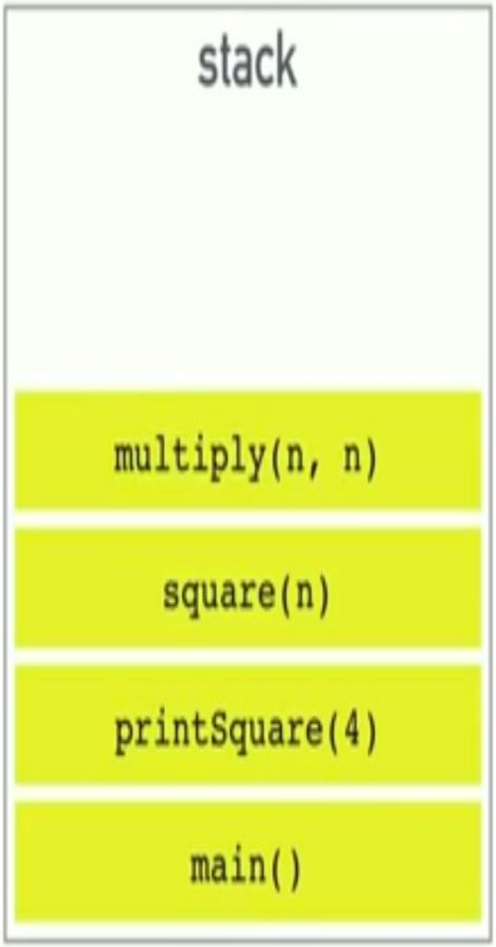
Call Stack


JS
Call Stack
function multiply(a, b) {
return a * b;
}
function square(n) {
return multiply(n, n);
}
function printSquare(n) {
var squared = square(n);
console.log(squared);
}
printSquare(4);
stack
multiply(n, n)
square(n)
printSquare(4)
main()


STACK ADDRESS
1000
100
96
92
88
84
80
76
72
68
64
60
56
52
PARAMETER 1
PARAMETER 2
RETURN ADDRESS
EBP
PARAMETER 3
PARAMETER 4
PARAMETER 5
RETURN ADDRESS
EBP
PARAMETER 6
PARAMETER 7
RETURN ADDRESS
EBP
100
110

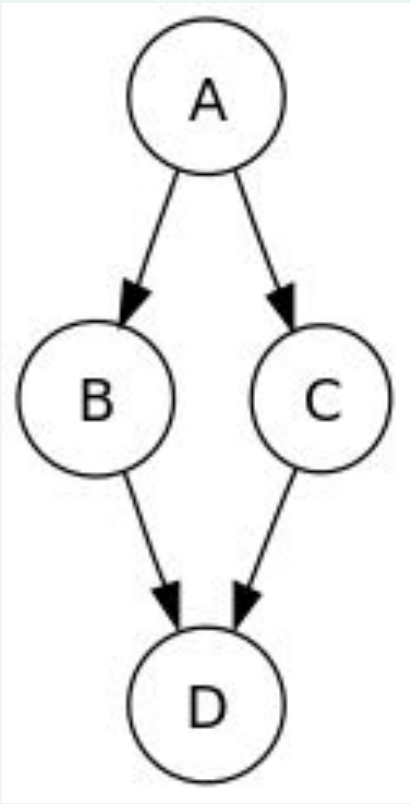
Call Stack

```
function multiply(a, b) {
    return a * b;
}

function square(n) {
    return multiply(n, n);
}

function printSquare(n) {
    var squared = square(n);
    console.log(squared);
}

printSquare(4);
```



Traversal Order

- A **preordering** is a list of the vertices in the order that they were first visited by the depth-first search algorithm.
  - E.g. A B D C / A C D B
- A **postordering** is a list of the vertices in the order that they were *last* visited by the algorithm.
  - E.g. D B C A / D C B A
- A **reverse postordering** is the reverse of a postordering, i.e. a list of the vertices in the opposite order of their last visit. Reverse postordering is not the same as preordering.
  - E.g A C B D / A B C D

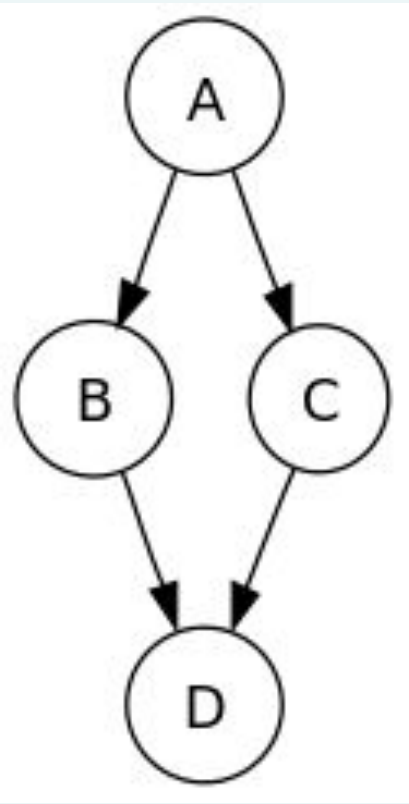


Depth First Search - DFS

```
1  procedure DFS-iterative(G,v):
2      let S be a stack
3      S.push(v)
4      while S is not empty
5            v = S.pop()
6            if v is not labeled as discovered:
7                label v as discovered
8                for all edges from v to w in G.adjacentEdges(v) do
9                    S.push(w)
```

DFS Iterative

- Visited order?
  - A B D C
  - A C D B



Breath First Search

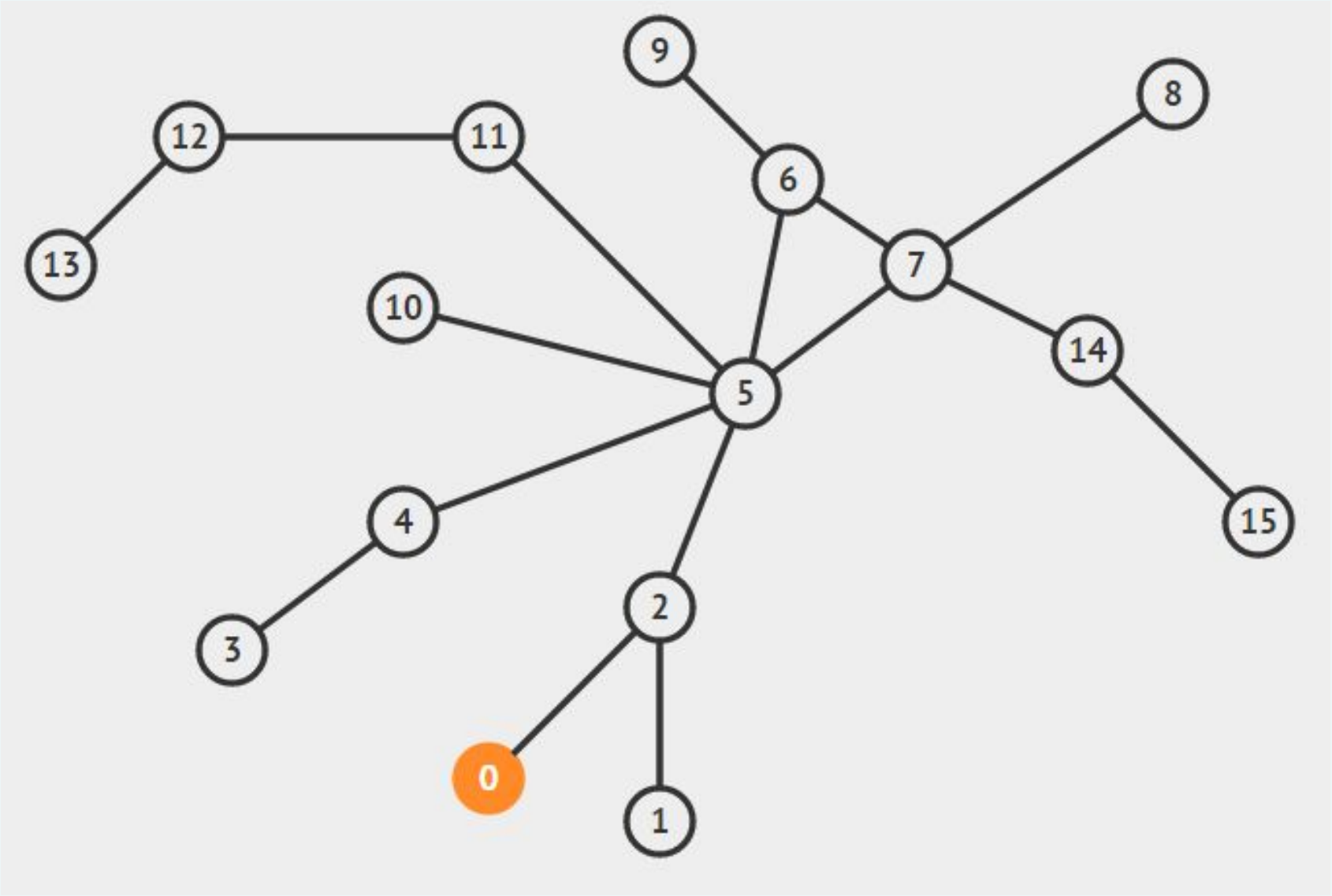
A breadth-first search (BFS) is another technique for traversing a finite graph. BFS visits the sibling vertices before visiting the child vertices

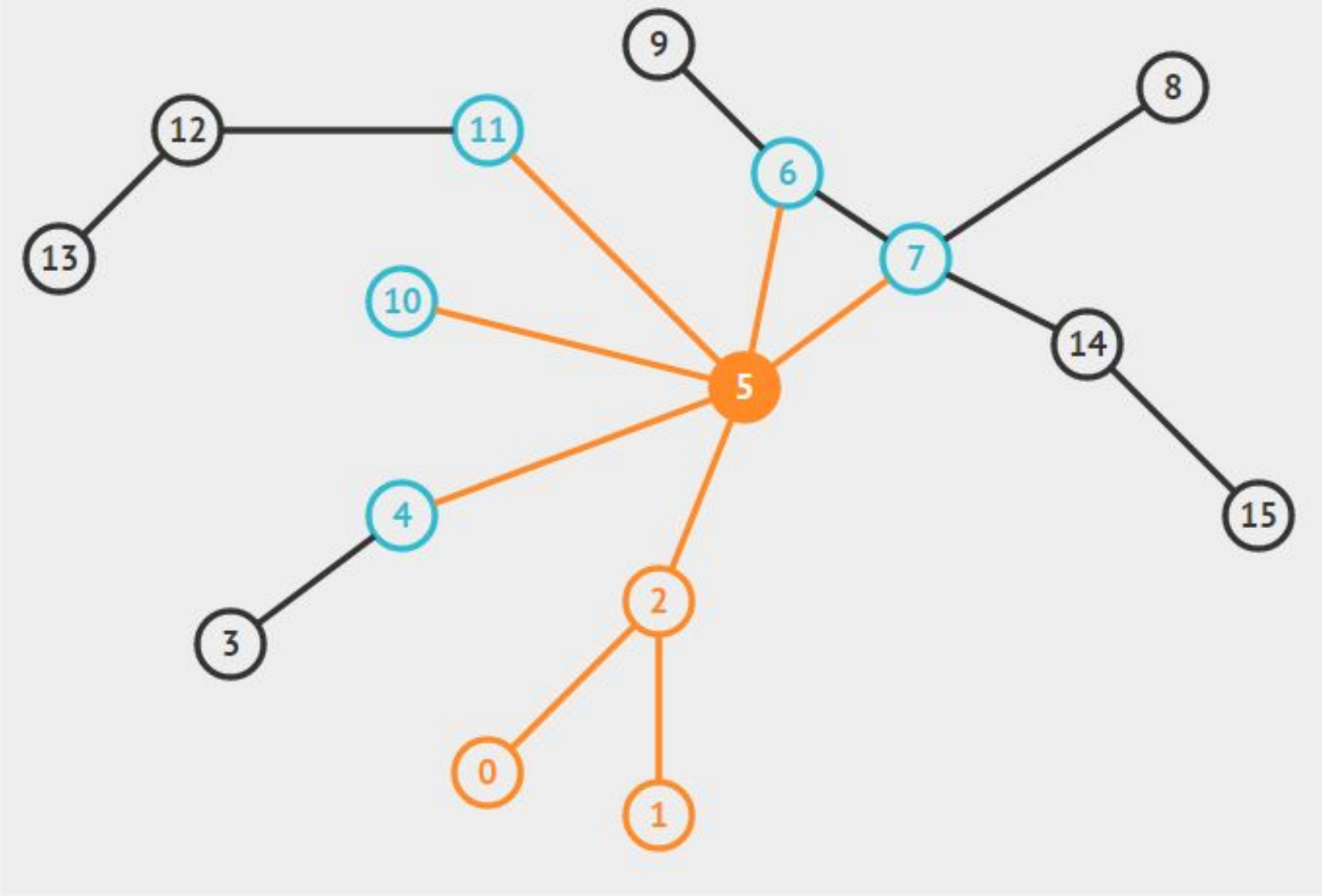
A queue is used in the search process. This algorithm is often used to find the shortest path from one vertex to another.

```
procedure BFS(G, v) is
    create a queue Q
    enqueue v onto Q
    mark v
    while Q is not empty do
        w ← Q.dequeue()

        for all edges e in G.adjacentEdges(w) do
            x ← G.adjacentVertex(w, e)
            if x is not marked then
                mark x
                enqueue x onto Q
```

```
procedure BFS(G, v) is
    create a queue Q
    enqueue v onto Q
    mark v
    while Q is not empty do
        w ← Q.dequeue()

        for all edges e in G.adjacentEdges(w) do
            x ← G.adjacentVertex(w, e)
            if x is not marked then
                mark x
                enqueue x onto Q
```

Breadth First Search




9
8
12
11
6
13
7
10
14
5
4
15
3
2
0
1

Breath First Search

```
 1 Breadth-First-Search(Graph, root):
 2
 3     for each node n in Graph:
 4         n.distance = INFINITY
 5         n.parent = NIL
 6
 7     create empty queue Q
 8
 9     root.distance = 0
10     Q.enqueue(root)
11
12     while Q is not empty:
13
14         current = Q.dequeue()
15
16         for each node n that is adjacent to current:
17             if n.distance == INFINITY:
18                 n.distance = current.distance + 1
19                 n.parent = current
20                 Q.enqueue(n)
```

Applications

- https://www.spoj.com/EIUPROGR/problems/dis2_02/
  - DFS Directed Graph
  - DFS Undirected Graph
  - BFS Directed Graph
  - BFS Undirected Graph
  - Thành phần liên thông
  - Thành phần liên thông 2

Complexity

- O(V+E)

- Each vertex is only visited once due to the fact that DFS will only recursively explore a vertex u if status[u] = unvisited — O(V)
- Every time a vertex is visited, all its k neighbors are explored and therefore after all vertices are visited, we have examined all E edges — (O(E) as the total number of neighbors of each vertex equals to E).

Backtracking

Backtracking is a general algorithm for finding all (or some) solutions to some computational problems, notably constraint satisfaction problems, that incrementally builds candidates to the solutions, and abandons each partial candidate *c* ("backtracks") as soon as it determines that *c* cannot possibly be completed to a valid solution.

- Ref:
  - https://en.wikipedia.org/wiki/Depth-first_search
  - https://en.wikipedia.org/wiki/Breadth-first_search
  - https://en.wikipedia.org/wiki/Backtracking
  - http://visualgo.net/dfsbfs.html