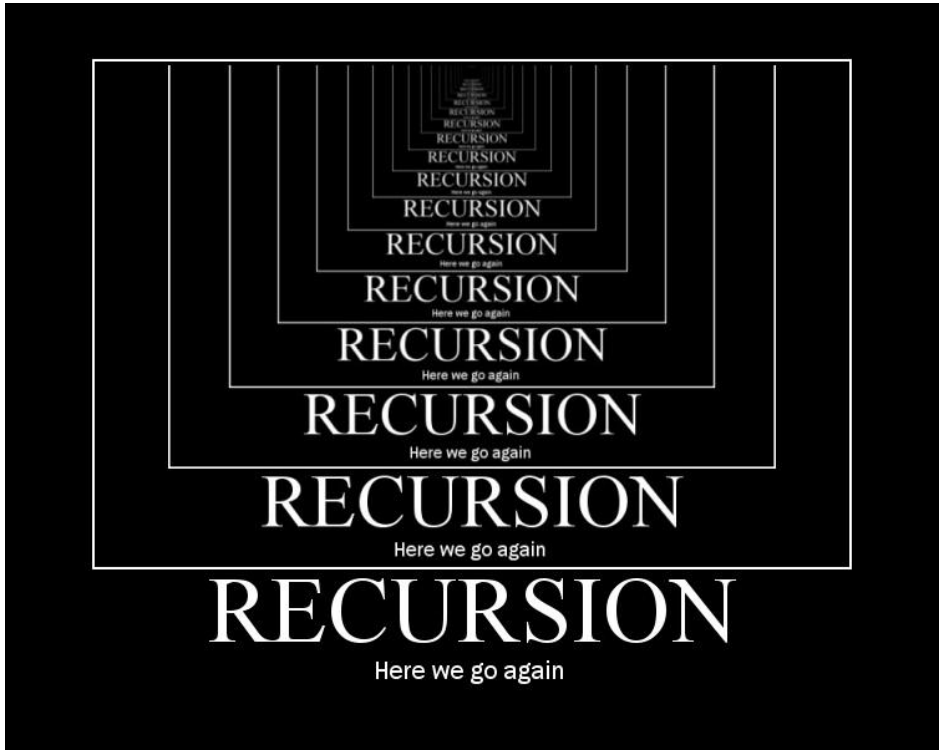
# Recursion + BFS/DFS

# Recursion

**Recursion** in computer science is a method where the solution to a problem depends on solutions **to smaller instances of the same problem** (as opposed to iteration). The approach can be applied to many types of problems, and recursion is one of the central ideas of computer science.

# Recursion


RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again

# Factorial - pseudocode

**function** factorial is:

**input**: integer *n* such that *n* >= 0

**output**: [*n* × (*n*-1) × (*n*-2) × … × 1]

1. if *n* is 0, **return** 1
2. otherwise, **return** [ *n* × factorial(*n*-1) ]

**end** factorial

# Factorial - example (4!)

$f_4 = 4 * f_3$

$= 4 * (3 * f_2)$

$= 4 * (3 * (2 * f_1))$

$= 4 * (3 * (2 * (1 * f_0)))$

$= 4 * (3 * (2 * (1 * 1)))$

$= 4 * (3 * (2 * 1))$

$= 4 * (3 * 2)$

$= 4 * 6$

$= 24$

# Factorial - Python code

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

# Other examples

Fibonacci sequence

Towers of Hanoi

Greatest common divider

Sudoku solver

System admin - finding and deleting files

# Recursion - binary trees

1

2

3

4

8

# Introduction to Graph Theory

Graph theory and in particular the graph ADT (abstract data-type) is widely explored and implemented in the field of Computer Science and Mathematics.

Consisting of vertices (nodes) and the edges (optionally directed/weighted) that connect them, the data-structure is effectively able to represent and solve many problem domains.

# Introduction to Graph Theory

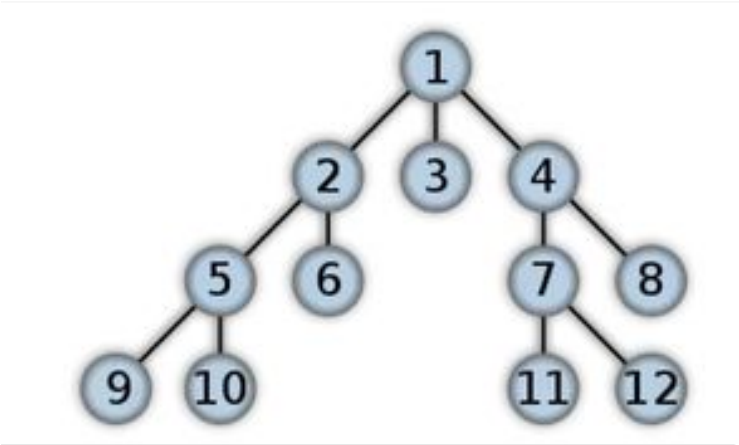
One of the most popular areas of algorithm design within this space is the problem of checking for the existence or (shortest) path between two or more vertices in the graph.

We will be exploring two of the simpler available algorithms, Depth-First and Breadth-First search to achieve the goals highlighted below:

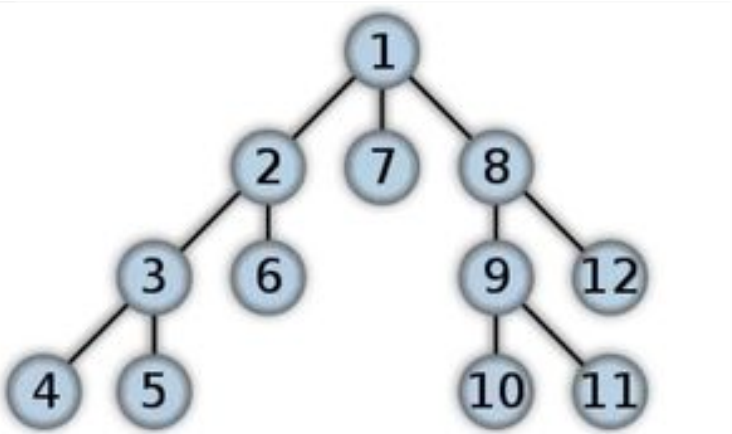
- Find all vertices in a subject vertices connected component.
- Return all available paths between two vertices.
- And in the case of BFS, return the shortest path (length measured by number of path edges).

# Traversing/searching trees

BFS




DFS



# DFS - Depth First Search

It’s implemented with stack (LIFO)

# DFS - Depth First Search

```
Set all nodes to "not visited";
s = new Stack(); ******* Change to use a stack
s.push(initial node); ***** Push() stores a value in a stack
while ( s ≠ empty ) do {
    x = s.pop(); ****** Pop() remove a value from the stack
    if ( x has not been visited ) {
        visited[x] = true; // Visit node x !
        for ( every edge (x, y) /* we are using all edges ! */ )
            if ( y has not been visited )
                s.push(y); ***** Use push() ! } }
```

# BFS - Breadth First Search

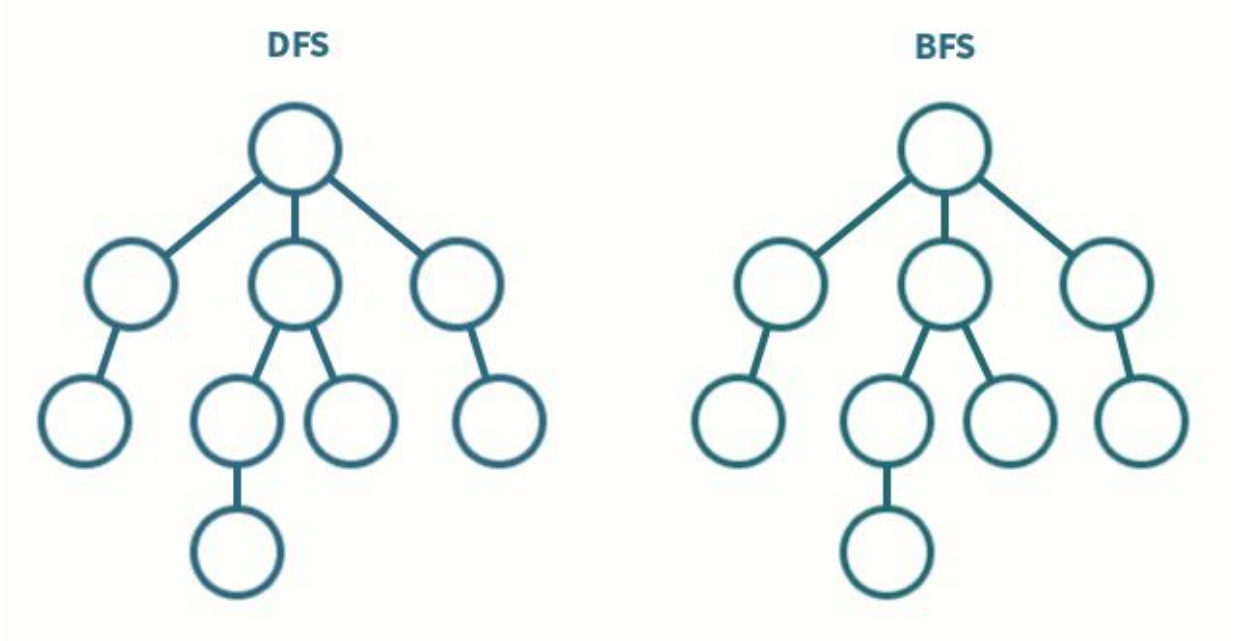
It’s implemented with queue (FIFO)

Same results as DFS but with the added guarantee to return the shortest-path first

# BFS - Breadth First Search pseudocode

```
Set all nodes to "not visited";
q = new Queue();
q.enqueue(initial node);
while ( q ≠ empty ) do {
    x = q.dequeue();
    if ( x has not been visited ) {
        visited[x] = true; // Visit node x !
        for ( every edge (x, y) /* we are using all edges ! */ )
        if ( y has not been visited )
            q.enqueue(y); // Use the edge (x,y) !!! } }
```

# Comparison of BFS and DFS



# Code example

http://hack.engeto.com/bfsdfs.py

(+ http://www.raph.ws/2014/02/bfs-and-dfs-with-python.html

+ http://eddmann.com/posts/depth-first-search-and-breadth-first-search-in-python/

+ http://eddmann.com/posts/depth-first-search-and-breadth-first-search-in-python/ )