# Introduction

# Problem

Find the best algorithm to determine if a graph is connected or not for both directed an undirected graph.

# Possible Algorithms

For each type of graph (directed and undirected) similar yet different algorithms need to be employed in order to ensure the analysis is complete and correct.

## Undirected Graphs

For undirected graphs, if there is a path between two nodes A and B then there is also a path between B and A, thus each combination of paths only needs to be explored in one direction. Thus, the following algorithm can be employed using some type of pathfinding algorithm in the inner loop:

for i from 0 to n:

for j from i + 1 to n:

if path does not exist between nodes i and j:

return false

return true

With this rough algorithm in place the next step is to find the most appropriate pathfinding algorithm for the given task. There are two main pathfinding algorithms that can be used in this scenario.

### Breadth-First Search

Breadth-First Search is an algorithm that explores a graph from a given node by exploring all the nodes adjacent to itself and then running that same algorithm recursively until its destination has been found. Breadth-First Search is known to always find the optimal path exists. The algorithm is usually employed as follows:[[1]](#footnote-1)

BFS (G, s)

let Q be queue.

Q.enqueue( s )

mark s as visited.

while ( Q is not empty)

v = Q.dequeue( )

for all neighbours w of v in Graph G

if w is not visited

Q.enqueue( w )

mark w as visited.

### Depth-First Search

In contrast to Breadth First Search, Depth First Search travels as deep as it can get until is reaches a terminal node and then goes up one level and runs the same process on all surrounding nodes until it has found a path. Due to the nature of this algorithm, it will always find a path if there is one, but it may not necessarily be optimal. The algorithm is as follows:[[2]](#footnote-2)

DFS-iterative (G, s):

let S be stack

S.push( s )

mark s as visited.

while ( S is not empty):

v = S.top( )

S.pop( )

for all neighbours w of v in Graph G:

if w is not visited :

S.push( w )

mark w as visited

## Directed Graphs

# Time Complexity Analysis

# Selected Algorithms

# Sample Graphs

# References

1. Prateek Garg, “Breadth First Search,” *HackerEarth*, accessed April 24, 2018, https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/. [↑](#footnote-ref-1)
2. Prateek Garg, “Depth First Search,” *HackerEarth*, accessed April 24, 2018, https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/. [↑](#footnote-ref-2)