Francesco Corso

4/27/18

CMPSC 360

# Introduction

This is a project for CMPSC 360 that is designed to test knowledge of algorithm complexity and connectivity of graphs. The goal is to create a program to find out if a graph is connected or not.

# Problem

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

# 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. In every case the following algorithm is used to get all the neighbors of each node:

getNeighbors(G, s)

let L be list

for every node in G

if the node is adjacent to s

L.push(node)

return L

## Undirected Graphs

### Breadth-First Search

Breadth-First Search (BFS) 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 iteratively until its destination has been found. 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.

return visited.size == G.nodes

### Depth-First Search

In contrast to Breadth First Search, Depth First Search (DFS) travels as deep as it can get until it reaches a terminal node and then goes up one level and runs the same process on all surrounding nodes until it has explored all nodes. 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

return visitited.size == G.nodes

## Directed Graphs

The same two algorithms above can be applied but it also requires a special loop to check if there is a path to every node from any starting point. Thus, the following algorithm can be applied:

checkDirected(G)

for every node in G

if all nodes cannot be reached (either BFS or DFS)

return false

return true

# Time Complexity Analysis

For the time complexity analysis, the number of comparisons performed by each algorithm will be considered for calculations.

## Breadth-First Search

For BFS the worst-case for the number of loops for the outer loop is because then the algorithm has had to check every node and add it to the explored set. Likewise, it the same as the inner loop because it also in the worst-case scenario. Each loop will run comparisons in order to check if the loop should execute. Within the outer loop, the getNeighbors algorithm will be run, which has a loop which will execute n times with comparisons for execution checking. Within the body of that loop, 1 comparison will be run thus meaning comparisons will be run in total every time that algorithm is run. This algorithm being nested in a loop that runs n times means that that algorithm will perform comparisons total. Within the inner loop there is one comparison meaning the loop has a total comparisons for a single iteration. With this loop also being nested in a loop that executes times meaning it performs a total of . By putting those two together and adding the final comparison that is returned at the end the algorithm runs a total of comparisons for this algorithm. Thus, the algorithm has a time complexity of .

## Depth-First Search

Since the iterative version of DFS uses the same structure except for swapping a queue for a stack, thus it will perform the exact same number of comparisons meaning it also has a time complexity of .

## Directed Adjustment

To apply the algorithm to a directed graph the outer loop will run at most times performing comparisons, thus the modified algorithm will perform comparisons which means the algorithm has a time complexity of .

## Directed Checking

In order to determine if the graph is directed or not since the algorithm for a directed graph has a higher time complexity it is better to check if a graph is directed and run the more efficient algorithm. The following algorithm can be used:

isDirected(G)

for every node in G

for every node in F

if node A has the same number of connections as node B

return true

return false

Each loop runs a total of times with comparisons to for execution checking. Within the body of the inner loop comparison is performed per iteration, meaning that a total of comparisons are performed meaning this algorithm has a time complexity of .

# Selected Algorithms

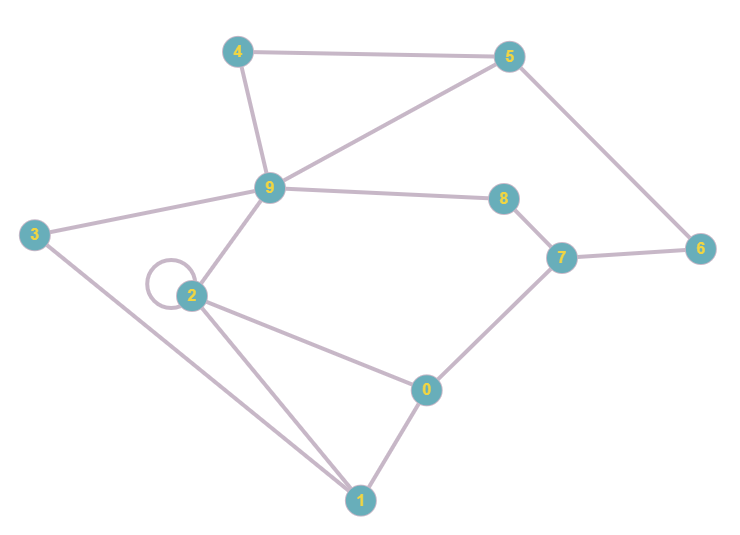
Due to the fact that both algorithms have the same time complexity. When adjusted for a directed graph, the algorithms still have the same time complexity relative to each other. Due to that fact, there is no inherent advantage to using one algorithms over the other. For this application, undirected graphs will use BFS and directed graphs will use DFS.

# Sample Graphs

The following four graphs were used to test the program in question. Each were all generated from a website, cited in both the references page and the footnote of this page.[[3]](#footnote-3)

## Example Graph 1:

The following graph is a connected undirected graph:



This graph can be represented by the following adjacency matrix:

The execution of the program was as follows:

Enter the number of nodes in the graph: 10

Enter row 1 of the matrix: 0 1 1 0 0 0 0 1 0 0

Enter row 2 of the matrix: 1 0 1 1 0 0 0 0 0 0

Enter row 3 of the matrix: 1 1 1 0 0 0 0 0 0 1

Enter row 4 of the matrix: 0 1 0 0 0 0 0 0 0 1

Enter row 5 of the matrix: 0 0 0 0 0 1 0 0 0 1

Enter row 6 of the matrix: 0 0 0 0 1 0 1 0 0 1

Enter row 7 of the matrix: 0 0 0 0 0 1 0 1 0 0

Enter row 8 of the matrix: 1 0 0 0 0 0 1 0 1 0

Enter row 9 of the matrix: 0 0 0 0 0 0 0 1 0 1

Enter row 10 of the matrix: 0 0 1 1 1 1 0 0 1 0

Matrix input complete!

Now computing connectivity...

Running Breadth-First Search on the undirected graph.

Start node: 0

Checking node 0

Checking node 1

Checking node 2

Checking node 7

Checking node 3

Checking node 9

Checking node 6

Checking node 8

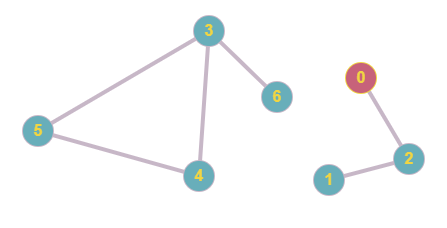
Checking node 4

Checking node 5

The graph is connected!

## Example Graph 2:

The following graph is an undirected graph that is not connected:



This graph can be represented by the following adjacency matrix:

The execution of the program was as follows:

Enter the number of nodes in the graph: 7

Enter row 1 of the matrix: 0 0 1 0 0 0 0

Enter row 2 of the matrix: 0 0 1 0 0 0 0

Enter row 3 of the matrix: 1 1 0 0 0 0 0

Enter row 4 of the matrix: 0 0 0 0 1 1 1

Enter row 5 of the matrix: 0 0 0 1 0 1 0

Enter row 6 of the matrix: 0 0 0 1 1 0 0

Enter row 7 of the matrix: 0 0 0 1 0 0 0

Matrix input complete!

Now computing connectivity...

Running Breadth-First Search on the undirected graph.

Start node: 0

Checking node 0

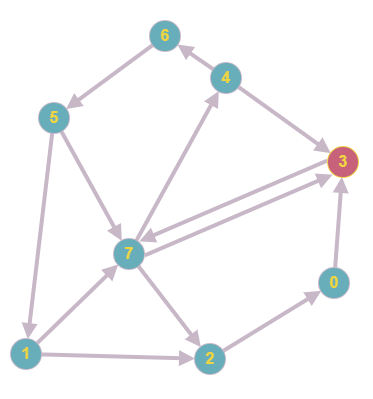
Checking node 2

Checking node 1

The graph is not connected...

## Example Graph 3:

The following graph is a directed graph that is connected:



This graph can be represented by the following adjacency matrix:

The execution of the program was as follows:

Enter the number of nodes in the graph: 8

Enter row 1 of the matrix: 0 0 0 1 0 0 0 0

Enter row 2 of the matrix: 0 0 1 0 0 0 0 1

Enter row 3 of the matrix: 1 0 0 0 0 0 0 0

Enter row 4 of the matrix: 0 0 0 0 0 0 0 1

Enter row 5 of the matrix: 0 0 0 1 0 0 1 0

Enter row 6 of the matrix: 0 1 0 0 0 0 0 1

Enter row 7 of the matrix: 0 0 0 0 0 1 0 0

Enter row 8 of the matrix: 0 0 1 1 1 0 0 0

Matrix input complete!

Now computing connectivity...

Running Depth-First Search on the directed graph.

Start node: 0

Checking node 0

Checking node 3

Checking node 7

Checking node 4

Checking node 6

Checking node 5

Checking node 1

Checking node 2

Start node: 1

Checking node 1

Checking node 7

Checking node 4

Checking node 6

Checking node 5

Checking node 3

Checking node 2

Checking node 0

Start node: 2

Checking node 2

Checking node 0

Checking node 3

Checking node 7

Checking node 4

Checking node 6

Checking node 5

Checking node 1

Start node: 3

Checking node 3

Checking node 7

Checking node 4

Checking node 6

Checking node 5

Checking node 1

Checking node 2

Checking node 0

Start node: 4

Checking node 4

Checking node 6

Checking node 5

Checking node 7

Checking node 2

Checking node 0

Checking node 1

Checking node 3

Start node: 5

Checking node 5

Checking node 7

Checking node 4

Checking node 6

Checking node 3

Checking node 2

Checking node 0

Checking node 1

Start node: 6

Checking node 6

Checking node 5

Checking node 7

Checking node 4

Checking node 3

Checking node 2

Checking node 0

Checking node 1

Start node: 7

Checking node 7

Checking node 4

Checking node 6

Checking node 5

Checking node 1

Checking node 3

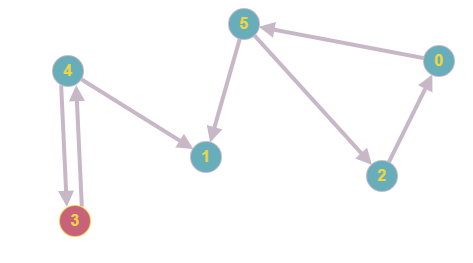
Checking node 2

Checking node 0

The graph is connected!

## Sample Graph 4

The following graph is a directed graph that is not connected:



This graph can be represented by the following adjacency matrix:

The execution of the program was as follows:

Enter the number of nodes in the graph: 6

Enter row 1 of the matrix: 0 0 0 0 0 1

Enter row 2 of the matrix: 0 0 0 0 0 0

Enter row 3 of the matrix: 1 0 0 0 0 0

Enter row 4 of the matrix: 0 0 0 0 1 0

Enter row 5 of the matrix: 0 1 0 1 0 0

Enter row 6 of the matrix: 0 1 1 0 0 0

Matrix input complete!

Now computing connectivity...

Running Depth-First Search on the directed graph.

Start node: 0

Checking node 0

Checking node 5

Checking node 2

Checking node 1

The graph is not connected...

# References

“Creating Graph from Adjacency Matrix,” *Graph Online*, accessed April 26, 2018, http://graphonline.ru/en/create\_graph\_by\_matrix.

Prateek Garg, “Breadth First Search,” *HackerEarth*, accessed April 24, 2018, https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/.

Prateek Garg, “Depth First Search,” *HackerEarth*, accessed April 24, 2018, https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/.

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)
3. “Creating Graph from Adjacency Matrix,” *Graph Online*, accessed April 26, 2018, http://graphonline.ru/en/create\_graph\_by\_matrix. [↑](#footnote-ref-3)