Longest Path Algorithm Pseudocode

This algorithm assumes that nodes and edges are kept track of. This is an algorithm for a weighted graph that allows looping. The weights for the graphs are turned into negative numbers.

# Part 1: Find all loops and keep track of them.

#### We will do this using the Graph Coloring method.

* Insert the edges into an adjacency list.
* Call the DFS function which uses the coloring method to mark the vertex.
* Whenever there is a partially visited vertex, backtrack till the current vertex is reached and mark all of them with cycle numbers. Once all the vertices are marked, increase the cycle number.
* Once DFS is completed, iterate for the edges and push the same marked number edges to another adjacency list.
* Iterate in another adjacency list and ~~print the vertex cycle-number wise~~. add loops to a list to access later.

Source: <https://www.geeksforgeeks.org/print-all-the-cycles-in-an-undirected-graph/>

# Part 2: Create a Maximum Spanning Tree

#### A Maximum Spanning Tree is created using Kruskal’s Algorithm applied to a negatively weighted graph.

KRUSKAL(G):

A = ∅

For each vertex v ∈ G.V:

MAKE-SET(v)

For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):

if FIND-SET(u) ≠ FIND-SET(v):

A = A ∪ {(u, v)}

UNION(u, v)

return A

Source: <https://www.programiz.com/dsa/kruskal-algorithm>

# Part 3: Use Breadth First Search to find path

1. Create a queue and a visited array initially filled with 0, of size V where V is the number of vertices.
2. Insert the starting node in the queue, i.e. push u in the queue and mark u as visited.
3. Run a loop until the queue is not empty.
4. Dequeue the front element of the queue. Iterate all its adjacent elements. If any of the adjacent elements is the destination return true. Push all the adjacent and unvisited vertices in the queue and mark them as visited.
5. Return false as the destination is not reached in BFS.

Source: <https://www.geeksforgeeks.org/find-if-there-is-a-path-between-two-vertices-in-a-given-graph/#:~:text=Approach%3A%20Either%20Breadth%20First%20Search,return%20true%20else%20return%20false>.

# Part 4: Adding a loop for extra measure.

#### If the destination node is adjacent to the source node, check for the longest loop of the source node to add to the path from source to destination that doesn’t include the destination node in the loop’s path.

If length of path from A to B is 2 (just A->B):

for each loop in the list:

if loop includes A, but not B and length of loop path is greater than previous found:

longestLoop = currentLoop

return longestLoop (which is a path)

insert this loop path into the correct place in the current path

add source node to beginning of loop path and destination node to the end