Longest Path Algorithm Research

This document describes the dilemmas of the “longest path problem” and how we will get around it to create our longest path algorithm for our routing protocol.

# Dijkstra’s Algorithm and OSPF

The Open Shortest Path First (OSPF) routing protocol uses Dijkstra’s algorithm to calculate the shortest path from one router to another. This algorithm works as follows:

Dijkstra’s algorithm finds a shortest path tree from a single source node, by building a set of nodes that have the minimum distance from the source.

The graph has the following:

* vertices, or nodes, denoted in the algorithm by v or u;
* weighted edges that connect two nodes: (u,v) denotes an edge, and w(u,v) denotes its weight.

This is done by initializing three values:

* dist, an array of distances from the source node s to each node in the graph, initialized the following way: dist(s) = 0; and for all other nodes v, dist(v) = ∞. This is done at the beginning because as the algorithm proceeds, the dist from the source to each node v in the graph will be recalculated and finalized when the shortest distance to v is found
* Q, a [queue](https://brilliant.org/wiki/queues-basic/) of all nodes in the graph. At the end of the algorithm's progress, Q will be empty.
* S, an empty [set](https://brilliant.org/wiki/sets/), to indicate which nodes the algorithm has visited. At the end of the algorithm's run, S will contain all the nodes of the graph.

The algorithm proceeds as follows:

1. While Q is not empty, pop the node v, that is not already in S, from Q with the smallest dist (v). In the first run, source node s will be chosen because dist(s) was initialized to 0. In the next run, the next node with the smallest dist value is chosen.
2. Add node v to S, to indicate that v has been visited
3. Update dist values of adjacent nodes of the current node v as follows: for each new adjacent node u.

* if dist (v) + weight(u,v) < dist (u), there is a new minimal distance found for u, so update dist (u) to the new minimal distance value;
* otherwise, no updates are made to dist (u).

(Dijkstra's Shortest Path Algorithm. Brilliant.org. Retrieved 14:15, October 18, 2020, from <https://brilliant.org/wiki/dijkstras-short-path-finder/>)

# What is the “Longest Path Problem”?

In [graph theory](https://en.wikipedia.org/wiki/Graph_theory) and [theoretical computer science](https://en.wikipedia.org/wiki/Theoretical_computer_science), the longest path problem is the problem of finding a simple [path](https://en.wikipedia.org/wiki/Path_(graph_theory)) of maximum length in a given graph. A path is called simple if it does not have any repeated vertices; the length of a path may either be measured by its number of edges, or (in [weighted graphs](https://en.wikipedia.org/wiki/Weighted_graph)) by the sum of the weights of its edges This is otherwise known as the “Travelling Salesman Problem” or the “Hamiltonian Cycle Problem”

The longest path problem is [NP-hard](https://en.wikipedia.org/wiki/NP-hard) and the decision version of the problem, which asks whether a path exists of at least some given length, is [NP-complete](https://en.wikipedia.org/wiki/NP-complete).

The longest path can be found of a graph that is directed and acyclic (has no cycles), but a computer network is not directed or strictly acyclic.

For this reason, our algorithm will not be truly the longest but “longish”:

Below are some ideas for specific algorithms and criteria the algorithm must meet.

## Algorithm Criteria

1. In the case of multiple existing paths, the path chosen must be longer than the “shortest path” possible.
2. No infinite looping
3. Utilize edge weights to make a decision.
4. Work on an undirected graph, potentially with cycles (A NETWORK IS AN UNDIRECTED GRAPH)

## Potential Algorithm Ideas

### How do we modify Dijkstra’s to find the longest path?

It seems like Dijkstra’s could work to find the longest path like so:

1. Select starting node as base node.
2. Compute path length to all neighbor nodes of the base node, which have not been the base node yet. If the path is longer than the longest one calculated up to now, the new path will be stored.
3. Select that node as the new base node, with the longest distance to the starting node and repeat step 2.
4. The optimal solution is found when there are no more paths left that haven’t been calculated.

However, in order for Dijkstra’s algorithm to work to find the longest path, the graph must be directed and without any loops. This is an issue because every edge in a network is bidirectional, so i cannot be directed, and a network layout is never guaranteed to be loop-free.

For this reason, using only dijkstra’s algorithm is not reasonable for a routing protocol.

Source: <https://www.wiwi.uni-kl.de/bisor-orwiki/Longest_paths:_Dijkstra_2>

Source:<https://stackoverflow.com/questions/20280671/longest-path-on-weighted-undirected-graph#:~:text=1%20Answer&text=If%20the%20graph%20is%20undirected,the%20graph%20must%20be%20directed>.

# So, What Can We Do?

### We need to remove loops, while maintaining connections to all nodes. Based on this criteria, we need to create a **Maximum Spanning tree!**

In general, people want to find a Minimum Spanning Tree, which is a subset of the edges of a [connected](https://en.wikipedia.org/wiki/Connected_graph), edge-weighted undirected graph that connects all the [vertices](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) together, without any [cycles](https://en.wikipedia.org/wiki/Cycle_(graph_theory)) and with the minimum possible total edge weight.

Source:<https://en.wikipedia.org/wiki/Minimum_spanning_tree#:~:text=A%20minimum%20spanning%20tree%20>

So, a maximum spanning tree is the same thing, but the graph is connected by maximum possible edge weight. This is done by negating the weights for each edge and applying Kruskal's algorithm. After the maximum spanning tree is found, then doing a breadth first search to find the path from the source node to the destination node, should be a longest path of sorts.

Source:<https://mathworld.wolfram.com/MaximumSpanningTree.html#:~:text=A%20maximum%20spanning%20tree%20is,the%20command%20FindSpanningTree%5Bg%5D>.

### Kruskal’s Algorithm is as follows:

It falls under a class of algorithms called [greedy algorithms](http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Greedy/greedyIntro.htm) that find the local optimum in the hopes of finding a global optimum.

We start from the edges with the lowest weight and keep adding edges until we reach our goal.

1. Sort all the edges from low weight to high
2. Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
3. Keep adding edges until we reach all vertices.

### How can we make the paths even longer?

Before applying Kruskal’s algorithm to the network graph, find all the loops existing in the graph and keep track of them as a list of nodes.

All cycles are found using the ***Graph Coloring Method:***

* Step 1: call DFS traversal for the graph which can color the vertices.
* Step 2: If a partially visited vertex is found, backtrack till the vertex is reached again and mark all vertices in the path with a counter which is cycle number.
* Step 3: After completion of traversal, iterate for cyclic edge and push them into a separate adjacency list.
* Step 4: Print the cycles number wise from the adjacency list.

Source: <https://www.tutorialspoint.com/print-all-the-cycles-in-an-undirected-graph-in-cplusplus>

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

Then, when finding the paths using breadth first search on the maximum spanning tree:

* 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.

**To loop or not to loop?**

Find all simple cycles, keep track of them (excluding cycles that are just like A->B->A, because all the nodes will have bidirectional edges, so every node and adjacent node would count as one). Because every edge is bidirectional, any path can become a loop