**Assumed Goal:**

Find the closest node defined by collective closeness:

*Collective closeness = shortest distance from every query word*

1. *Distance from every node is considered for avoiding the greedy based algorithm from being biased to any single node.*
2. *Which one is to be considered close: Nodes having connection to all query words with large distance or nodes with less distance but not connecting to all nodes?*
3. *Multiple weight for the same relationship (same node, same relationship direction)*
4. *Many things could be done easily with neo4j cypher query, but I chose to do it myself with python.*

**Solution 1:**

**Minimum spanning weighted tree with Prim’s algorithm**

A brutal force approach for finding the closed node for a given list of queries.

Steps:

1. Build a minimum weighted spanning tree (only for query words present in the knowledge graph)

2. From the created tree, find the sum of the distance of every word with the query words. For unconnected node, add a constant large value.

For e.g.:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | e | f | G | h | i | J |
| a+b+c+d |  |  |  |  |  |  |

# TODO: Changing cost function to average (a+b+c+d)

3. Select 5 nodes having the list value of cost function (a+b+c+d)

**Pros:**

1. Could provide the optimized solution.

**Cons:**

1.It need to process each and every node in the graph.

2. Takes more space and time. Time Complexity O(nd). Where n= no. Of nodes and d= no. of query words in the list.

**Solution-2:**

**Neighbor based recursive greedy algorithm**

Solution 1 computes the shortest path for every node present in the knowledge graph. It is absurd to compute the shortest path for every node as the closest node lies in the neighborhood of the query node. It may be a good way to find an optimal solution in smaller knowledge graph. But for larger graph, it is computationally expensive and inefficient.

Steps:

1. Find the collective distance of each neighbor of all query words and sort them in an ascending order.
2. Create a list of n nodes (n: top n close nodes from the query words). Select a max distance: nth node. (i.e. collective distance of the nth node).
3. Expand all neighbors of the nodes expect nth node. If the collective distance of that node is less than the max distance, remove the nth node and add the new node to the list.
4. Repeat the process till no new neighbors with collective distance less than max distance is found

**Solution-3:**

**Closeness Centrality based algorithm**

**Solution-4:**

**Page Rank algorithm (accounting for relationship degree)**

Ranking could be an important metrics which accounts for the relationship degree of the nodes.