I collaborated with: W3023339

Problem
There's a natural intuition that two nodes that are far apart in a communication network—separated by many hops—have a more tenuous connection than two nodes that are close together. There are a number of algorithmic results that are based to some extent on different ways of making this notion precise. Here's one that involves the susceptibility of paths to the deletion of nodes.
Suppose that an n-node undirected graph $G = (V, E)$ contains two nodes s and t such that the distance between s and t is strictly greater than $n/2$. Show that there must exist some node v, not equal to either s or t, such that deleting v from G destroys all $s - t$ paths. In other words, the graph obtained from G by deleting v contains no path from s to t. Give an algorithm with running time $O(m + n)$ to find such a node v. Describe your algorithm in prose and prove that it works correctly. Make sure your algorithmic description is clear and concise. As a hint, imagine performing a breadth-first search from s. How large is each layer L_i along the way to t?
Solution
Since the distance between s and t is strictly greater than $n/2$, then the majority of the nodes of G have to be on the path from s to t. Consequently, a minority of the nodes of G are not on the path from s to t. In a BFS representation of G starting from s we will get that at least $n/2\ + 1$ nodes are on the path from s to t, and at most $n/2\ - 1$ nodes are not on the path from s to t. This means that there must be a layer of the path that is of size one, since the path has the majority of the nodes. The node in this layer is v, because removing it will disconnect G.

Algorithm 1 Finds v given a valid G

Require: n-node undirected graph $G = (V, E)$ contains two nodes s and t such that the distance between s and t is strictly greater than $n/2$.

```
1: Mark all $v \in V$ as unvisited
2: Let $T$ be an empty graph
3: Add $r$ to $T$; mark $r$ as visited; $r.level \leftarrow 0$
4: $currentLevel \leftarrow 0$
5: $levelCount \leftarrow 0$
6: Let $candidate$ be a node
7: while There are $visited$ vertices do
8:     $current \leftarrow$ some $visited$ vertex having minimum level
9:     Mark $current$ as explored
10:    for $unvisited$ neighbors $v$ of $current$ do
11:        Add $\{current, v\}$ to $T$
12:        Mark $v$ as $visited$
13:        $v.level \leftarrow current.level + 1$
14:        if $currentLevel$ does not equal $v.level$ then
15:            if $levelCount$ equals 1 then
16:                return $candidate$
17:            else
18:                $levelCount \leftarrow 1$
19:                $candidate \leftarrow v$
20:                $currentLevel \leftarrow v.level$
21:            end if
22:        else
23:            $levelCount \leftarrow levelCount + 1$
24:        end if
25:    end for
26: end while
```

This algorithm uses BFS to traverse G and form its representation in T. When adding a node to T, my algorithm checks if this node is the first node added of its level. If it is, then it checks the total number of nodes in the previous

level. If there is only one node in the previous level, then that node is returned. If there were multiple nodes in the previous level then we start counting the number of nodes in the current level and repeat the process.
BFS will reach every node in G, assuming it is a connected graph, so the potential node v will be reached. When adding a node in the level that follows $v.level$, we will see that $v.level$ contains v and only v. This is recognized as the note whose deletion from G destroys all $s - t$ paths.