Some practice counting

Question 1. Let $G = (V, E)$ be an undirected graph with n vertices and m edges. Show that if every vertex has degree at least $n/2$, the graph is connected. You may assume that n is even. As a hint, think about cuts in the graph. A cut (S, T) is a partition of the vertices into two sets S and T such that $S \cup T = V$ and $S \cap T = \emptyset$.

Question 2 (K&T 3.9)**.** 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?

BFS, again

Question 3. Consider the following algorithm:

Algorithm 1 BFSQ

Require: An undirected graph $G = (V, E)$ and a starting node $s \in V$.

```
1:  Initialize Q to be a queue with a single element s
2:  Initialize T to an empty tree
3:  Initialize all nodes to UNVISITED
4:  while Q is not empty do
5:      Remove the first node u from Q
6:      Mark u as VISITED
7:      for all edges (u, v) ∈ E adjacent to u do
8:          if v is UNVISITED then
9:              Add v to the end of Q
10:             Add (u, v) to T
11:         end if
12:     end for
13: end while
14: return T
```

(a) Prove that algorithm BFSQ produces a tree whose levels are the same as the L_i produced by the BFS algorithm from class. (*Hint*: show that if $u \in L_i$ and $v \in L_j$ with $i < j$, then u is added to Q before v.)

(b) Describe a representation for T which allows an arbitrary edge (u, v) to be added to T in $O(1)$ time.

Question 4. Consider the family of undirected graphs $\mathcal{H}_k$ defined as follows. $\mathcal{H}_k$ has 2^k vertices labelled with the integers 0 through $2^k - 1$. Vertices u and v are connected by an edge if and only if the binary

representations of u and v differ in exactly one bit. For example, in H_4, the vertices 5 and 13 are connected by an edge since $5 = 0101_2$ and $13 = 1101_2$ differ in exactly one bit.

Consider doing a BFS in $\mathcal{H}_{10}$ starting at node 0. How many vertices are in L_6, that is, the sixth layer generated by the BFS? Give your answer together with either a proof, or the program you used to calculate the answer. Either approach will receive full credit. (***Hint*** if you choose to write a program: to flip the `j`th bit of an integer `n`, use `n ^ (1 << j)`, that is, the bitwise XOR of `n` with 2^j.)

Graph diameters

Question 5. As in the text, we say that the *distance* between two nodes u and v in a graph $G = (V, E)$ is the minimum number of edges in a path joining them; we'll denote this by $dist(u, v)$. We say that the *diameter* of G is the maximum distance between any pair of nodes, that is,

$$diam(G) = \max_{u,v \in V} dist(u, v).$$

Give an $O(n + m)$-time algorithm to find the diameter of a **tree** $T = (V, E)$. Prove that your algorithm is correct.

Question 6. Does your algorithm from Question 5 work for **all** undirected graphs? If so, prove it. If not, provide a counterexample.

Question 7.

- On a scale of 1 to 10, how difficult was this assignment?
- How many hours would you estimate that you spent on it?
- Which was your favorite question? Your least favorite?

Extra credit: Approximating the diameter of a graph

Question 8. There's a good chance your linear-time algorithm in Question 5 doesn't extend to undirected graphs. (If it does, you almost certainly have a STOC publication. Congratulations!) It turns out that, until very recently, we didn't have any method for computing the diameter of a graph that didn't first compute the shortest path between all pairs of nodes. When graphs are dense, all-pairs shortest paths is fairly expensive, so some people have explored quicker algorithms which *estimate* the diameter of the graph. Develop a linear-time algorithm that, given a graph G, returns a diameter estimate that is always within a factor of 1/2 of the true diameter. That is, if the true diameter is $diam(G)$ then you should return a value k where $diam(G)/2 \leq k \leq diam(G)$.