Avi Hirsch

Design and Analysis of Algorithms

HW #8: KT 3.6, 3.7, 3.9

3.6. *Prove that "if T is both a depth-first search tree and a breadth-first search tree rooted at u, then G cannot contain any edges that do not belong to T."*

Proof by contradiction/contraposition: If G *does* contain an edge that does not belong to T, then T cannot be both a depth-first search tree and a breadth-first search tree. If it was, then the two nodes that form that edge must be at most one level off in T (since it's a BFS tree), and one node of that edge must an ancestor of the other in T (since it is a DFS tree). Thus, the ancestor must necessarily be its descendant's parent, and the edge connecting them must therefore be in T; since it is not, we have a contradiction, and the original statement must be true.

3.7. *Claim: Let G be a graph on n nodes, where n is an even number. If every node of G has degree at least n/2, then G is connected.*

True. Proof by contradiction: If G is *not* connected, then since there are at least two connected components, the smallest one must contain at most half of the nodes (in a case where there are 2 connected components with the same number of nodes in each, i.e., n/2 nodes). The degree of every node in this smallest connected component can be at most one less than the number of nodes in that component (since it itself is one element of that connected component), or n/2 – 1, which is less than n/2 and thus contradicts our initial assumption.

3.9. *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.*

We know that the distance from s to t is at least n/2, or half the number of nodes in G. In other words, there is no shorter path than a path of length n/2 to get from s to t. A BFS starting at node s will find the shortest path to each node from s. Upon reaching t, we know that at least n/2 nodes have been reached to get to t, and therefore there are at least n/2 "layers" between s and t in that BFS.

At least one of these layers must contain only one node v, since if all the layers had at least 2 nodes, there would have to be at least n nodes between s and t (2*n/2), which cannot be, since neither s nor t are included in that set of nodes. Finally, by the properties of BFS, removing this one-node layer would sever the path between s and t, since that node will always be reached when moving from s to t.

Thus, to find this node, run a BFS from s, keeping track of the nodes reached and their layers, and node v will be in its own layer.