Shortest Unweighted Path

Remember that the (unweighted) length of a path in a graph is the number of edges the path contains. Consider a directed graph and let S be any specific node in this graph. We now give an algorithm for finding the shortest path from S to every other node in the graph.

The algorithm maintains a queue of nodes, which initially contains only S. It gives each node a value, which ultimately will be the length of the shortest path. Finally, it gives each node a predecessor node in its path from S.

Initially make the value of each node except S be "INFINITY". If you are thinking of a Java implementation, INFINITY can be either Integer.MAX_VALUE or Double.MAX_VALUE, depending on how you want to think of the values. Make the value of S be 0.

Now perform the following steps until the queue is empty.

a) Remove the head of the queue. Call this node X.
b) For each outgoing edge from X to another node Y, if the value of Y is INFINITY, make the new value of Y be the value of X + 1, make the predecessor of Y be X, and add Y to the queue.

Now, how do we know this algorithm works?

I claim that if node X has distance n from S then the value this algorithm assigns to X is n. This is certainly true when n is 0 or 1. For other nodes let $S=X_0$->X_1->X_2->...->X_n=X be a path of length n to X. Suppose node X_t is the first node on this path that the algorithm assigns the wrong distance to. This means that node X_{t-1} has the correct distance. When X_{t-1} is removed from the queue and assigned the distance t-1, X_t will be added to the queue with distance t. So X_t in fact gets the correct distance. This means that all of the nodes in the path, including X, get the correct distance.

Note that this algorithm visits every edge in the graph (at least every edge that is reachable from S and so has running time O(|V|).