Longest Paths

We have looked at 3 shortest path algorithms:

The algorithm for unweighted graphs uses a queue so that we put into the queue first all paths of length 1, then all paths of length 2 to nodes we haven't reached yet, then all paths of length 3 to nodes we haven't reached yet, and so forth.

Dijkstra's algorithm for graphs with non-negative weights on the edges uses a priority queue. When a node comes out of the priority queue we know the cheapest path to it and we add to the priority queue all endpoints of its outgoing edges that haven't themselves come out of the priority queue.

The Bellman-Ford algorithm works for any kind of weights. It uses a queue like the unweighted algorithm, though when something emerges from the queue we only know the tentative cheapest path to it. We look at the destinations of its outgoing edges and add back to the queue any vertex that this gives a cheaper path to.

Now suppose we want to find the longest path through a graph rather than the shortest path. Will any of those three algorithms convert to a longest path algorithm by ‘reversing the inequality”
i. e. replacing < with >

Answer: the Bellman-Ford algorithm converts easily into a longest path algorithm; the others don't convert at all.

The unweighted paths algorithm and Dijkstra's algorithm both depend on knowing the best path to a node when we remove it from the queue or priority queue. We can't possibly know the longest path to the first node that comes out of the queue without exploring the rest of the edges, so those algorithms will not convert to longest path algorithms. On the other hand, the Bellman-Ford algorithm always works in terms of the "best path so far'. When we pull node X out of the queue and see that X has an edge to vertex Y that gives a *better* path to Y we update the information from Y and add Y to the queue. "Better" can mean shorter or it can mean longer; it just depends on what kind of path you are trying to find.

Bellman-Ford is the most expensive of the path-finding algorithms in terms of running time, but it is also the most versatile. It can handle weighted and unweighted graphs (for unweighted ones think of every edge as having weight 1), and for weighted graphs it can handle both positive and negative weights.