Sketch of Breadth-First and Depth-First Search

1. Breadth-First Search (First In First Out)

(a) Select a vertex \( v \), give it the empty predecessor label \( p(v) = - \) and the distance label \( d(v) = 0 \), and place it in a queue.

(b) While the queue is not empty, remove vertex \( u \) from the queue. For each unlabeled neighbor \( w \) of \( u \), give it the predecessor label \( p(w) = u \) and the distance label \( d(w) = d(u) + 1 \), and place it in the queue.

When the queue is empty, all the vertices in the component containing \( v \) have been labeled. For each such vertex \( u \), \( d(u) \) is the distance (the length of the shortest path) from \( v \) to \( u \), and a path of that length can be found by working backwards from \( u \) : \( u, p(u), p^2(u), \ldots \).

2. Depth-First Search (Last In First Out)

(a) Select a vertex \( v \), give it the empty predecessor label \( p(v) = - \) and place it in a stack.

(b) While the stack is not empty, examine the top vertex \( u \) of the stack. If \( u \) has no unlabeled neighbors, then remove it from the stack. If \( u \) has at least one unlabeled neighbor, choose one unlabeled neighbor \( w \), give it the predecessor label \( p(w) = u \) and the distance label \( d(w) = d(u) + 1 \), and place it in the stack.

When the stack is empty, all the vertices in the component containing \( v \) have been labeled. For each such vertex \( u \), \( d(u) \) is the length of a path from \( v \) to \( u \) (but not necessarily the shortest path), and a path of that length can be found from \( v \) to \( u \) by working backwards from \( u \) : \( u, p(u), p^2(u), \ldots \).

Note that with either algorithm, you can detect whether or not the component containing \( v \) has any cycles—while processing the vertex \( u \) in step (b), if an already labeled neighbor, say \( w \), other than \( p(u) \) is discovered, then you can trace paths back from \( u \) and \( w \) to a common vertex. These two paths, together with the edge \( uw \), form a cycle. If no such neighbor is found, then the component has no cycle, and is seen to have \( n \) vertices and \( n - 1 \) edges from the construction. If some such neighbor and cycle are found, then the component is seen to have \( n \) vertices and strictly more than \( n - 1 \) edges.