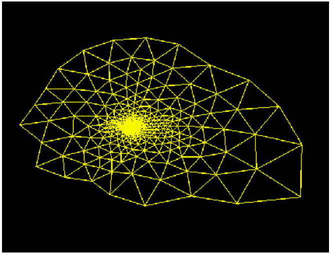
CI583: Data Structures and Operating Systems
Algorithmic strategies to solve NP-hard problems



Depth-first and breadth-first

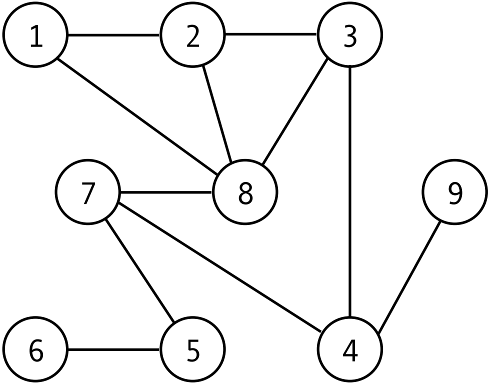
Many NP problems can be reduced to a problem in which the nodes in a graph must be visited exactly once – this is a more general idea of traversal, which we used with trees.

For instance, we may need to transmit a network packet to every computer on a network, making sure that no computer receives it twice.

There are two ways we might do this: depth-first and breadth-first.

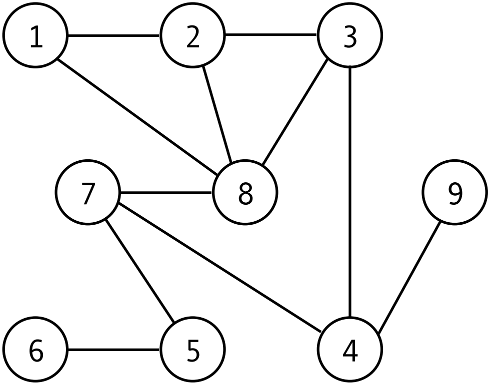
Depth-first and breadth-first

A depth-first traversal will go as far as possible down a given path before it considers any other. A breadth-first traversal goes evenly in many directions.



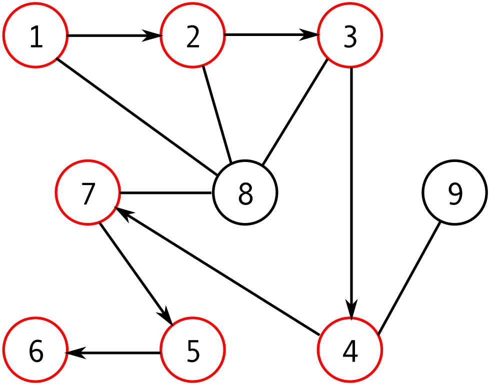
Depth-first traversal

In a depth-first traversal, we visit the starting node then follow edges through the graph until we reach a dead-end. In an undirected graph a node is a dead end if all nodes adjacent to it have been visited. In a directed graph a node is also a dead end if it has no outgoing edges.



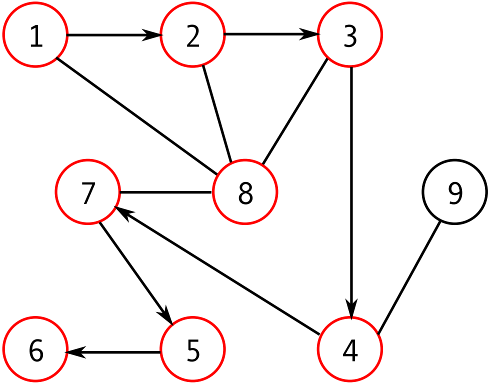
Depth-first traversal

When we reach a dead end we go back up the path until we find a node with an unvisited adjacent node. One traversal of this graph starting at 1 reaches a dead end at 6: [1, 2, 3, 4, 7, 5, 6].



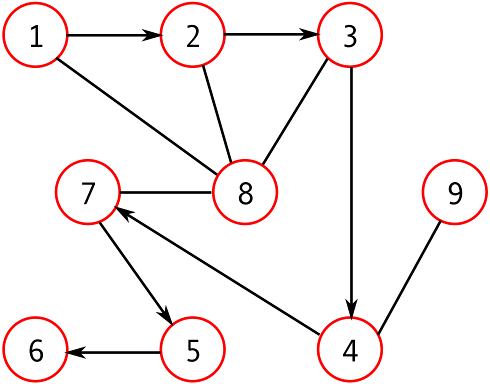
Depth-first traversal

We then go back to node 7 and find that 8 is unvisited. We visit 8 and reach a dead end.



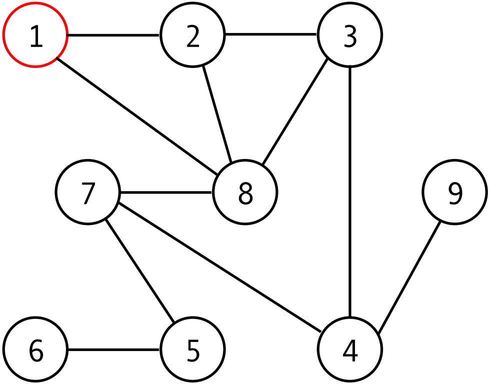
Depth-first traversal

We then go back to 4 and find that 9 is unvisited. The next time we go back up the path we end up at the starting node and we are done.



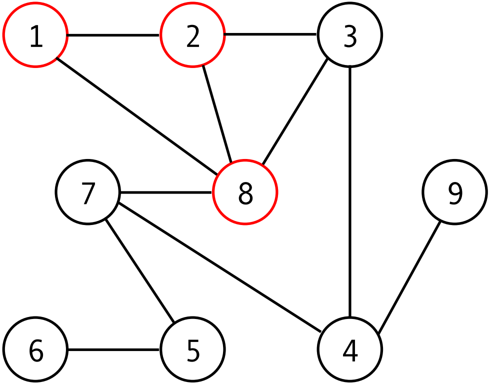
Breadth-first traversal

With a breadth-first traversal we first visit all nodes which are adjacent to the starting node. Then we visit all nodes which are two edges away from the starting node, and so on. For the graph below, one traversal starting at 1 is [1, 2, 8, 3, 7, 4, 5, 9, 6].



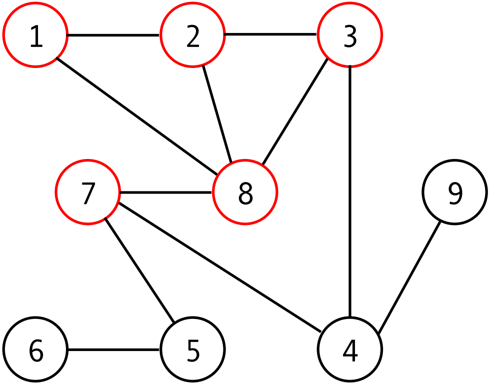
Breadth-first traversal

With a breadth-first traversal we first visit all nodes which are adjacent to the starting node. Then we visit all nodes which are two edges away from the starting node, and so on. For the graph below, one traversal starting at 1 is [1, 2, 8, 3, 7, 4, 5, 9, 6].



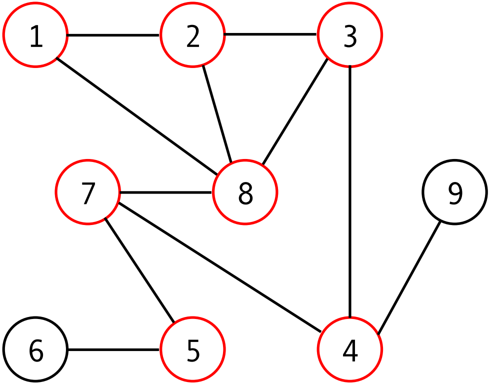
Breadth-first traversal

With a breadth-first traversal we first visit all nodes which are adjacent to the starting node. Then we visit all nodes which are two edges away from the starting node, and so on. For the graph below, one traversal starting at 1 is [1, 2, 8, 3, 7, 4, 5, 9, 6].



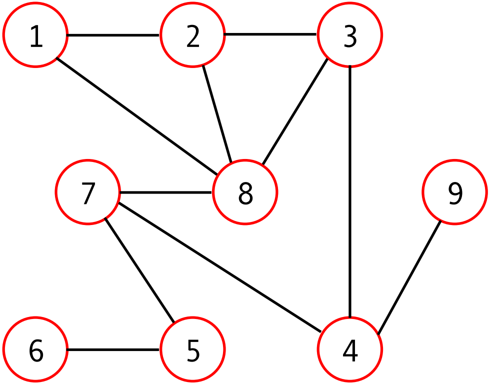
Breadth-first traversal

With a breadth-first traversal we first visit all nodes which are adjacent to the starting node. Then we visit all nodes which are two edges away from the starting node, and so on. For the graph below, one traversal starting at 1 is [1, 2, 8, 3, 7, 4, 5, 9, 6].



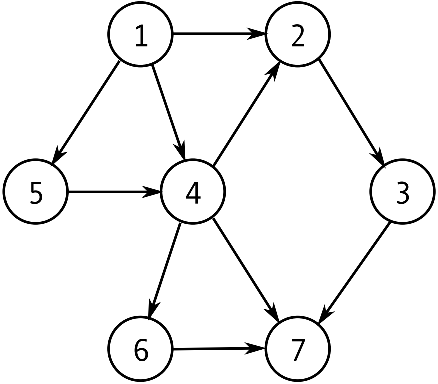
Breadth-first traversal

With a breadth-first traversal we first visit all nodes which are adjacent to the starting node. Then we visit all nodes which are two edges away from the starting node, and so on. For the graph below, one traversal starting at 1 is [1, 2, 8, 3, 7, 4, 5, 9, 6].



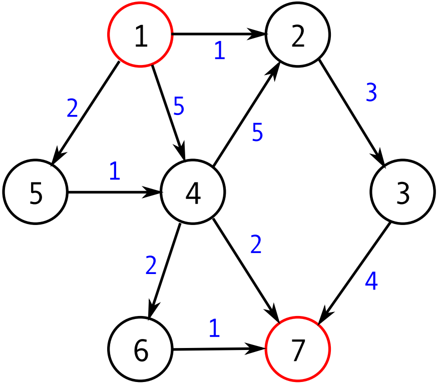
Traversal

Implementations of depth-first traversal often use a stack. Implementations of breadth-first traversal often use a queue. Give DF and BF traversals starting at the node labelled 1 for this directed graph:



Weighted graphs

Consider the problem of finding “cheapest” paths in a directed and weighted graph:



Minimum spanning trees

The problem of finding cheapest or shortest paths in a a weighted and connected graph reduces to that of finding the minimum spanning tree (MST). A spanning tree for a graph, G, contains all the nodes of G and a subset of the edges and has no cycles. A *connected* graph is one in which there is a path from every node to any other.

The MST for a graph, G, is a spanning tree in which the total of the edge weights is minimal.

We can calculate the MST by brute force – for n edges this takes 2^n comparisons between potential MSTs – this is an NP problem.