Cycles in an Undirected Graph

1. Determine if an undirected graph $G = (V, E)$ is connected, where $|V| = n$ and $|E| = m$.

2. Find all the cycles in an undirected graph G.

3. (Optional) Find all articulation points in G.

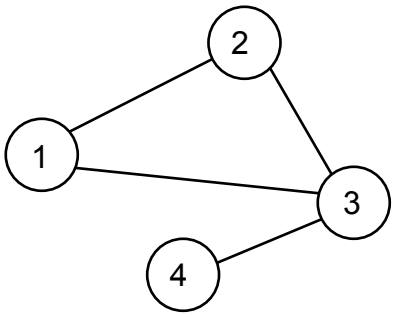
Terminology: Given an undirected graph, a depth-first search (DFS) algorithm constructs a directed tree from the root (first node in the V). If there exists a directed path in the tree from v to w, then v is an predecessor of w and w is a descendant of v.

A node-node adjacency structure is an $n \times n$ matrix such that entry $a_{ij} = 1$ if node i is adjacent to node j and 0 otherwise. A node-edge adjacency structure lists for each node, the nodes adjacent to it.

Example

Let $V = \{1,2,3,4\}$ and $E = \{(1,2), (1,3), (2,3), (3,4)\}$. The node-node adjacency structure is

0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0



The node-edge adjacency structure is

1: 2,3
2: 1,3
3: 1,2,4
4: 3

Note that node 4 is called a *leaf*.

(1 and 2) Depth-first search (DFS) can be used to solved all three problems. It is usually assumed that the graph data structure is of the type node-edge adjacency instead of node-node adjacency. In the node-edge adjacency structure, the nodes are numbered from 1 to n. The node-i record lists the nodes adjacent to node i (connected to node i by an edge).

DFS starts by setting node 1 as the current node.

DFS iteration (i is the current node)

– If one or more nodes of the node-i record were not yet visited from i, let node j be the first node not visited. If node j was already visited by DFS (obviously from node other than node i), mark edge (i,j) as *back* edge otherwise mark edge (i,j) as *tree* edge and set i as parent of j. Mark node j as visited in the node-i record. Set node j as current node.

– Otherwise (all nodes of node-*i* record were visited), set node *k* as current node, where node *k* is the parent of *i*.

The algorithm stops when node 1 is current (i.e., the algorithm returns to node 1). If all nodes were visited, then the graph is connected. Each back edge (*i*,*j*) defines a cycle. A cycle consists of the back edge (*i*,*j*) and unique tree edges forming the path from *j* to *i*. The cycles so defined by the back edges form a cycle base of the graph (see below). Every cycle of the graph is the union (exclusive OR) of two or more cycles from this cycle base. Of course, the number of cycles in a graph can be exponential in the number of nodes of the graph.

For the above example the DFS goes as follows:

current node	edge	
1	(1,2)	tree, 1 is parent of 2
2	(2,3)	tree, 2 is parent of 3
3	(3,4)	tree, 3 is parent of 4
4	none	
3	(3,1)	back
3	none	
2	none	
1	END	

Note that a subroutine is needed to find the union of two or more cycles so that all the cycles can be uncovered. One way (I think) to find all the cycles is to restart the algorithm at each node. However, this would be inefficient because many cycles already identified would be rediscovered when restarting.

Cycle base and cycle representation

A cycle base is a set of $m - (n - 1)$ cycles that are independent in the sense that we cannot reconstruct one cycle from the set by the union (defined below) of two or more other cycles of the set. The set of $m - (n - 1)$ back edges defines a cycle base. This is not the only cycle base. Actually there can be an exponential number of them.

(Note: The method will find non-existing cycles in case there are no overlapping edges at all between the two cycles in base. This can be fixed by first applying AND on each two cycles in base.)

The number of back edges is always $m - (n - 1)$ provided the graph is connected. This is because the maximum number of edges in a tree is $(n - 1)$, as in a spanning tree, and in the DFS algorithm the tree edges form a spanning tree. This is easily proved by showing that the tree edges induce a connected subgraph that contains every node of the graph and contains no cycle. If the graph is not connected then the number of back edges is $m - (n - 1) + (p - 1)$ where p is the number of connected components.

Every other cycle of the graph can be obtained by the union of two or more cycles of the cycle base. By union I mean an "exclusive OR" operation. To perform this operation, we represent cycles by edge-incidence vectors. Consider a second example:

node 1: 2
node 2: 1,3,6
node 3: 2,4,6
node 4: 3,5,6
node 5: 4,6
node 6: 2,3,4,5

Note, node 1 is a leaf.

The DFS algorithm will produce the following result (I omitted some steps).

edge (1,2): tree, 1 is parent of 2
edge (2,3): tree, 2 is parent of 3
edge (3,4): tree, 3 is parent of 4
edge (4,5): tree, 4 is parent of 5
edge (5,6): tree, 5 is parent of 6
edge (6,2): back
edge (6,3): back
edge (6,4): back

A cycle consists of the back edge (*i*,*j*) and the unique tree edges forming the path from *j* to *i*. The cycles that would be identified are:

2-3-4-5-6-2
3-4-5-6-3
4-5-6-4

The cycles that would be missed are:

2-3-6-2
2-3-4-6-2
3-4-6-3

Now assume that the vector of all edges in the graph is

(1,2) (2,3) (2,6) (3,4) (3,6) (4,5) (4,6) (5,6)

The incidence vector of cycle (2,3,4,5,6,2) is 01110101 and the one of cycle (3,4,5,6,3) is 00011101. An exclusive OR on these two incidence vectors gives the vector 01101000 which corresponds to cycle (2,3,6,2). If more than two cycles of the base are involved, then the operation is similar. I am not sure exactly what happens when the operation is performed on a set of cycles of the base for which one or more edges appear more than twice. Do we count the edge if it appears an odd number of times? Maybe any simple cycle can be retrieved by adding a set of cycles of the base such that no edge appears more than twice in the set. But I am not sure.

3. Articulation points

Node $k \in V$ is an articulation point of G if $G \setminus \{k\}$ has more than one component. That is, when k is removed from the graph, one or more unconnected subgraphs result. In the above example, node 3 is an articulation point.

To uncover articulation points with DFS you need to record one extra label at every node. This label is usually denoted by low(v), where low(v) = min{v, w : (u,w) is a back edge with $u \neq v$ a descendant of v and w a predecessor of v}. At the end of the algorithm the nodes with low(v) = v are articulation points if v has at least one child (this condition rules out leaves). The exception is node 1. Node 1 is an articulation point if and only if it is the parent of two or more nodes.

The expression "min{v,w}" simply means that low(v) is equal to v itself except if there exists a back edge that connects a successor (descendant) of v to a predecessor (ascendant) of v, say w. In this case the value of low(v) = w. Note that "min{v,w}" does not mean that you select the minimum value of v or w. The DFS algorithm constructs a directed tree rooted at node 1. This tree determines a partial ordering of the nodes. A node w is a successor of another node v (and v is a predecessor of w) if there exists a directed path from v to w in the tree.

In practice, a portion of the DFS algorithm is a numbering scheme. We number (or renumber) the nodes in the order that they are "discovered" by the DFS. Thus if DFS "discovers" a back edge (u,w) we need to update the low(v) labels on all internal nodes on the unique path of tree edges from w to u. A brute force method of doing this is, starting from u, traverse the tree edges backwards until w is reached. If this brute force method is applied every time a back edge is "discovered", the whole algorithm is clearly $O(mn)$. To make it $O(n)$, one uses a trick proposed by Tarjan but I don't recall what it is.