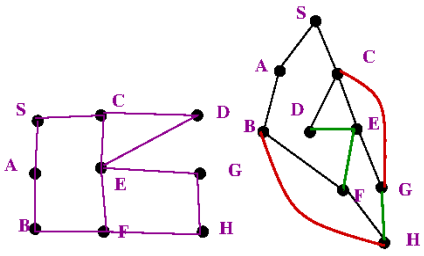
**Tutorial 9**

1. Please study the undirected graph below and the tree edges (in black), non-tree edges in green (and some forbidden edges in red). We have seen that this gives a tree, in fact, a shortest-path tree for the vertex s. In other words, it is a tree T for which the path from s to another vertex u is the shortest path in the original graph. Thus bfs produces a shortest path tree. Note that there may be many shortest path trees and the particular tree produced by bfs is one of them, which depends on the vertex ordering, i.e., how the vertices are ordered among themselves. Now, given a shortest path tree T, is there always an ordering of the vertices which will produce this tree T as outputs?




2. For the same graph below, execute dfs and produce a trace of the algorithm. List the vertices in order of push and pop events and mark the arrival and departure numbers. Modify the DFS to label tree edges twice with the counter, once while going forward, i.e., push (i.e., the recursive call) and the second time when popping, i.e., when exiting.

3. Suppose that there is an undirected graph G(V,E) where the edges are colored either red or blue. Given two vertices u and v. It is desired to (i) find the shortest path irrespective of colour, (ii) find the shortest path, and of these paths, the one with the fewest red edges, (iii) a path with the fewest red edges. Draw an example where the above three paths are distinct. Clearly, to solve (i), BFS is the answer. How will you design algorithms for (ii) and (iii)?

4. Suppose that we have a graph and we have run dfs starting at a vertex s and obtained a dfs tree. Next, suppose that we add an edge (u,v). Under what conditions will the dfs tree change? Give examples.

5. We have a new search algorithm which uses a set S for which we have two functions (i) add(x,S) which adds x to S, and (ii) y=select(S) which returns an element of S following a certain rule.

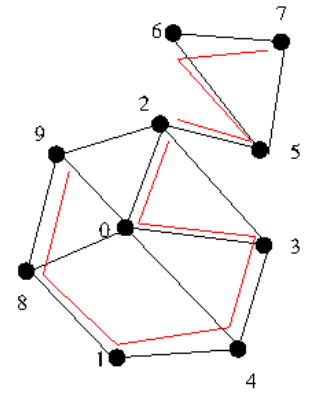
   Function mysearch
   Global visited;
   For all u visited[u]=false;
   for all edges e, found[e]=false;
   S=empty;

```
add(s,S); nos=1; record[nos]=s;
While nonempty(S)
   y=select(S)
   nos=nos+1; record[nos]=y;
   For all v adjacent to y
     If visited[v]==false
        visited[v]=true;
        found[(u,v)]=true;
        add(v,S);
     Endif;
   Endfor;
Endwhile
Endfunction;
```

(i) Compare the bfs and dfs algorithms with the above code. Take special care in understanding visited.
(ii) Let us look at the sequence record[1],record[2],...,record[n]. Show that there is a path from record[i] to record[i+1] using only edges which have been found at that point.
(iii) Compare BFS and DFS in terms of the above path lengths.

6. This is the theoretical basis of edge-2-connectedness. Let G(V,E) be a graph. We define a relation on edges as follows: two edges e and f are related (denoted by e~f iff there is a cycle containing both. Show that this is an equivalence relation. The equivalence class [e] of an edge e is called its 2-connected component.

7. Given a dfs tree for an undirected graph and a vertex v, we define C(v) as the edge (x,y) where x is a descendent of y while x is a parent of v, with x!=v and y!=v. Modify dfs to list C(v) and to compute |C(v)| . In the example below C(6)=1, C(5)=0 and C(1) =3. How much time does your algorithm take? Modify the code counterdfs.c




dfs=[2 0 3 4 1 8 9 5 6 7]