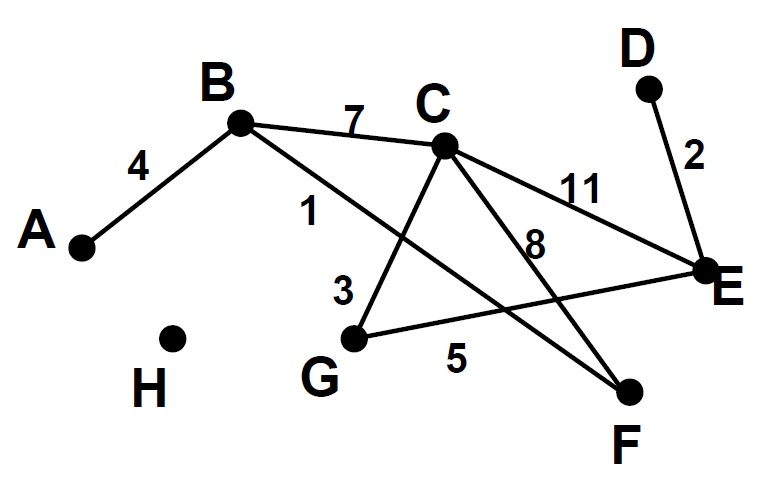
**Depth First Search Algorithm**

A depth-first traversal proceeds by moving from the last visited vertex to an adjacent unvisited one. If no unvisited adjacent vertex is available, the traversal backtracks to a previously visited vertex which does have an unvisited adjacent vertex. When we have backtracked all the way to the starting vertex and no further adjacent vertices remain unvisited, the algorithm terminates.



For the example graph, the following is a DFT starting at A:

A, B, C, E, D, G, F

Note that we do not visit the disconnected vertex H.

If we wanted to perform a DFT that includes all vertices, we could restart the algorithm with an unvisited vertex.

Note the implicit use of a stack (here, the call stack to manage the recursion).

The following will perform a DFT of the vertices in the same connected component as the starting vertex:

dfs(G=(V,E), a starting vertex s)

create empty stack L

L.push(s)

while (L.notEmpty)

v = L.pop()

if (v not yet visited)

visit(v)

for each vertex w, adjacent to v

if (w not yet visited)

L.push(w)

The efficiency class of this algorithm depends on which graph representation is used.

For an adjacency matrix, we have (|V 2|), and for adjacency list, we have (|V | + |E|).

The intuition behind this is that for each vertex in the graph, we have to visit each incident edge. With an adjacency matrix, this involves looking at each of |V | matrix entries (including those which are null, indicating that such an edge does not exist). For the adjacency list, we have the exact list of edges readily available, so across all vertices, we visit \_(|E|) total edges.