Graph Traversals: Scanning a graph or traversing it.

Two famous algorithms: Depth-first search (DFS) and Breadth-first search (BFS).

Both algorithms work for both directed and undirected graphs, though we focus more on undirected graphs.

DFS: the traversal is started from an arbitrary vertex $v$, which is called the root of DFS. The root is marked as visited. An arbitrary (unmarked) vertex $v_i$ connected to $v$ is then picked and a DFS starting from $v_i$ is performed recursively. The recursion stops when it reaches a vertex $v$ such that all the vertices connected to $v$ are already marked. If after the DFS from $v_i$ terminates all the vertices connected to $v$ are already marked, then DFS for $v$ terminates. Otherwise another arbitrary unmarked vertex $v_j$ connected to $v$ is picked, a DFS starting from $v_j$ is performed, and so on.

Algorithm DFS $(G,v)$:

begin
  mark $v$; DFS_number$[v]$ = DFSN; DFSN++; 
  Perform pre-work on $v$;
  for all edges $(v,w)$ do 
    if $w$ is unmarked then DFS $(G,w)$; add edge $(v,w)$ to tree DFS_tree;
    perform post-work for $(v,w)$
  end;
end;

To incorporate different applications, we do pre-work at the time the vertex is marked and post-work after we back track from a edge or find that the edge leads to a marked vertex.

Thus, if $G$ is connected then all vertices are marked and all edges visited.

Pf: Proof is by contradiction; Let $U$ be the set of unmarked vertices at the end.

Since $G$ is connected at least one vertex $u$ from $U$ is connected to a marked vertex $v$ and thus we should visit $u$ when we visit $v$ by definition of the algorithm.

Now since all vertices are visited and since whenever a vertex is visited all its edges are considered, all edges are considered too.

Thus the above application of DFS for checking connectivity of the graph is the number of marked vertices that we can count should be equal to $n$. DFS has other applications as well.
BFS traverses the graph level by level. If we start from a vertex $v$, then all $v$'s "children" are visited first. The second level includes a visit to all the "grandchildren", and so on. The procedure is non-recursive.

**Algorithm BFS($G$, $V$)**

begin
  mark $v$; $BFS\_number[v] = BFSN$; $level[v] = 0$;
  put $v$ in a queue (First In First Out);
  while the queue is not empty do
    remove the first vertex $w$ from the queue;
    perform pre work on $w$;
    for all edges $(w, x)$ such that $x$ is unmarked do
      mark $x$; $BFS\_number[x] = BFSN$; $BFSN++$; $level[x] = level[w] + 1$;
      add $(w, x)$ to the tree $BFS\_tree$;
      put $x$ in the queue;
  end;

Note that in both DFS and BFS, we assign a $DFS\_Number$ or $BFS\_Number$ to each vertex and we create a $DFS\_tree$ or a $BFS\_tree$.

**Thm:** For each vertex $w$, the path from the root to $w$ in the $BFS\_tree$ is a shortest path from the root to $w$ in $G$.

Proof is by induction on the level of a vertex.

**Thm:** If $(v, w)$ is an edge of $G$ not belonging to $T$, then it connects two vertices whose level numbers differ by at most 1.

**PF:** Consider the first of $v$ and $w$ which is visited and the other vertex has difference at most one (potentially zero).

The above thing is the main property of $BFS\_trees$.

**Thm (the main property of $DFS\_trees$):** Every edge in $G$ not belonging to $T$, connects two vertices of $G$, one of which is the ancestor of the other in $T$ (i.e., no cross edges).

**PF:** Let $(v, w)$ be an edge of $G$, and suppose that $v$ is visited by $DFS$ before $w$. After $v$ is marked, we perform $DFS$ starting from all neighbors. Since $u$ is a neighbor of $v$, $DFS$ either starts from $u$ in which case $(v, w)$ belongs to $T$, or the $DFS$ will visit $u$ before it backtracks from $v$, in which case $u$ is a descendant of $v$ in $T$.

→ See the properties of $DFS$ in directed graphs in the book.