Depth-First Search

Idea: Explore graph from a given vertex by first exploring all children of that vertex. To avoid looping through cycles, mark each vertex upon first visiting it; do not explore children of a previously visited vertex.

{Induce a depth-first forest on a graph.}
DFS(G)
    for v in vertices[G] do
        color[v] ← unexplored
        pred[v] ← nil
    for v in vertices [G] do
        if color[v] = unexplored then
            DFS-Visit(v)

{Induce a depth-first tree on a graph starting at v}
DFS-Visit(v)
    color[v] ← frontier
    for a in Adj[v] do
        if color[a] = unexplored then
            pred[a] = v
            DFS-Visit(a)
    color[v] ← processed

Note: DFS effectively uses a stack to process frontier vertices, in contrast with the queue used by BFS.

Analysis:

- DFS-Visit called exactly once on each vertex: Θ(V).
- Each directed edge explore exactly once in for loop within DFS-Visit: Θ(E).
- Total: Θ(V + E)
Edge Classification

- **Tree edges** are edges \((\text{pred}[v], v)\) forming depth-first forest.
- **Back edges** connect vertex to an ancestor in a depth-first tree.
- **Forward edges** are non-tree edges connecting vertex to a descendent in a depth-first tree.
- **Cross edges** are non-tree edges connecting (1) two vertices in a tree that are not in an ancestor/descendant relationship or (2) two vertices from different trees.

A tree is acyclic if there are no back edges.

Can mark edges by type during DFS by noting color of vertex when first encountered:

- **unexplored** indicates a tree edge
- **frontier** indicates a back edge
- **processed** indicates a forward or cross edge (can use timestamps -- see below -- to distinguish)
Timestamps

Can extend the simple DFS above to timestamp each discovery and finish step using a global clock:

{Induce a depth-first forest on a graph.}

\[
\text{DFS}(G) \\
\text{for } v \text{ in } \text{vertices}[G] \text{ do} \\
\quad \text{color}[v] \leftarrow \text{unexplored} \\
\quad \text{pred}[v] \leftarrow \text{nil} \\
\quad \text{time} \leftarrow 0 \quad \{\text{Assume time is a global variable}\} \\
\text{for } v \text{ in } \text{vertices } [G] \text{ do} \\
\quad \text{if color}[v] = \text{unexplored} \text{ then} \\
\quad \quad \text{DFS-Visit}(v)
\]

{Induce a depth-first tree on a graph starting at v}

\[
\text{DFS-Visit}(v) \\
\quad \text{color}[v] \leftarrow \text{frontier} \\
\quad \text{time} \leftarrow \text{time} + 1 \\
\quad \text{discovery}[v] \leftarrow \text{time} \\
\text{for } a \text{ in } \text{Adj}[v] \text{ do} \\
\quad \text{if color}[a] = \text{unexplored} \text{ then} \\
\quad \quad \text{pred}[a] = v \\
\quad \quad \text{DFS-Visit}(a) \\
\quad \text{color}[v] \leftarrow \text{processed} \\
\quad \text{time} \leftarrow \text{time} + 1 \\
\quad \text{finish}[v] \leftarrow \text{time}
\]

Parenthesis Theorem

For two vertices a and b in a depth-first forest of G, exactly one of the following three holds:

- The intervals (discovery\[a\], finish\[a\]) and (discovery\[b\], finish\[b\]) are disjoint.
- The interval (discovery\[a\], finish\[a\]) is nested within (discovery\[b\], finish\[b\])
  (True when a is a descendant of b in depth-first forest.)
- The interval (discovery\[b\], finish\[b\]) is nested within (discovery\[a\], finish\[a\])
  (True when b is a descendant of a in depth-first forest.)

Unexplored-path Theorem (CLR’s White-path Theorem)

In a depth-first forest of G, vertex d is a descendant of ancestor a iff at time
discovery\[a\], d can be reached from a along a path consisting entirely of unexplored
vertices.
Topological Sort

A directed acyclic graph (DAG) is a directed graph without cycles. A topological sort of a DAG $G = (V, E)$ is a linear ordering of vertices in $V$ consistent with the partial order $a < b$ if $(a, b) \in E$. In other words, each vertex in a topological sort must precede all its descendants in the DAG and must follow all of its ancestors.

**Approach 1:**

Modify DFS so that when it finishes processing a vertex, it prepends it to the front of an initially-empty global list. Since processing of a vertex is finished only when all descendents are finished, each vertex precedes all its descendants in the list. The running time is $\Theta(V + E)$ since DFS takes $\Theta(V + E)$ time.

**Approach 2:**

The in-degree of a vertex $v$ in a directed graph is the number of edges whose target is $v$.

```
Topological-Sort-2 (G)
L <- Empty-List
while vertices[G] ≠ {} do
  v <- Find-Vertex-With-Indegree=0(G)
  L <- Postpend(v, L)
  for e in Out-Edges(G, v)
    G <- Remove-Edge(e, G)
return L
```

Each vertex clearly follows all its ancestors. The running time can be made $O(V + E)$ (left as an exercise: see CLR 23.4-5 on p. 488).
**Connected and Strongly Connected Components**

A **connected component** of a graph is a maximal set of vertices such that for any two vertices \(a\) and \(b\) in the set, there is a path from \(a\) to \(b\) or from \(b\) to \(a\). In other words, two vertices are in the same connected component if there is a path from one to the other.

The connected components of a graph are computed by DFS: each tree in the depth-first forest is a connected component of the graph. Running time is running time of DFS = \(\Theta(V + E)\)

A **strongly connected component** of a graph is a maximal set of vertices such that for any two vertices \(a\) and \(b\) in the set, there is a path from \(a\) to \(b\) and from \(b\) to \(a\). In other words, in a strongly connected component, there is a path from every member of the set to every other member of the set.

The transpose of a graph \(G\), written \(G^T\), is a graph with the same vertices as \(G\) in which the directions of all edges have been reversed.

\[
\text{Strongly-Connected-Components}(G) \\
1. \text{Call DFS}(G) \text{ to compute } \text{finish}[v] \text{ for each vertex in } G \\
2. \text{Call Modified-DFS}(G^T), \text{ where the main loop of Modified-DFS processes vertices in order of decreasing } \text{finish}[v]. \\
3. \text{Each tree in depth-first forest of Modified-DFS}(G^T) \text{ is a strongly connected component of } G
\]

Running time is running time of DFS = \(\Theta(V + E)\).