DFS:

procedure previsit ( v ):
    pre ( v ) = counter; counter ++

procedure postvisit ( v ):
    post ( v ) = counter; counter ++

explore ( v )

visited ( v ) = true (pre)

for each edge ( v, u ) ∈ E

    if ( not visited ( u ) ) explore ( v ) (post)
DFS visits the entire graph:

for all $v \in V$: visited($v$) = false

for all $v \in V$:

if ( not visited ($v$) : explore ($v$) )

“connected components of a graph” “forest”

Previsit and Postvisit Orderings

previsit : moment of first discovery of a node

postvisit : moment of final departure
DFS in a directed graph
Different kinds of edges

Tree edges part of DFS forest
Forward edges lead to a descendant
Back edges lead to an ancestor
Cross edges lead neither to an ancestor nor a descendant (they lead to a node that has been post-visited)

• u is an ancestor of v

\[ \text{pre}(u) < \text{pre}(v) < \text{post}(v) < \text{post}(u) \]

```
[   [    ]       ]
u v v u
1 3 4 6
```

“means” “forward or tree” edge

• u is a descendant of v

\[ \text{pre}(v) < \text{pre}(u) < \text{post}(u) < \text{post}(v) \]

```
[   [    ]       ]
v u u v
```

• u is a cross edge

```
[   ]   [   ]
v v u u
```

```
8, 9 13, 14
```
Strongly Connected Components
Breadth first search

procedure bfs(G, s)
Input: Graph G = (V, E), directed or undirected; vertex s ∈ V
Output: For all vertices u reachable from s, dist(u) is set to the distance from s to u.

for all u ∈ V:
    dist(u) = ∞

dist(s) = 0
Q = [s] (queue containing just s)
while Q is not empty:
    u = eject(Q)
    for all edges (u, v) ∈ E:
        if dist(v) = ∞:
            inject(Q, v)
            dist(v) = dist(u) + 1
Shortest Paths
procedure dijkstra(G, s)
Input: Graph G = (V, E), directed or undirected;
positive edge lengths (u, v) ∈ E: vertex s ∈ V
Output: For all vertices u reachable from s, dist(u) is set
to the distance from s to u.

for all u ∈ V:
    dist(u) = ∞
    prev(u) = nil
    dist(s) = 0

H = makequeue(V) (using dist-values as keys)
while H is not empty:
    u = deletemin(H)
    for all edges (u, v) ∈ E:
        if dist(v) > dist(u) + d(u, v):
            dist(v) = dist(u) + d(u, v)
            prev(v) = u
            decreasekey(H, v)