Depth-first Search of a Digraph
(implmented using a stack)

Global variables and initializations:

$S$: A stack of vertices. The vertices on $S$ will be the vertices on the path from the starting vertex up to and including the current vertex.

$adjList$: An array of lists. $adjList[v]$ is the adjacency list of vertex $v$. For simplicity we allow the adjacency lists to be destroyed by the algorithm. $removeElement(adjList[v])$ will remove an element from $adjList[v]$ and return it.

$color$: an array of vertices. $color[v]$ will be the color of vertex $v$. Initially $color[v] = white$ for all vertices.

$time$: relative time, initially 0.

d$: an array of vertices. $d[v]$ will be the “discover time” of vertex $v$, i.e., the time at which $v$ is pushed onto the stack, colored gray, and preprocessed.

$f$: an array of vertices. $f[v]$ will be the “finish time” of vertex $v$, i.e., the time at which $v$ is postprocessed, colored black, and popped from the stack.
**Depth first search of entire graph:** Perform depth-first search from some white vertex, then from a second white vertex, a third, etc., until no white vertices remain.

```c
void depthFirst(Digraph G)
    Initialize variables as above;
    for ( each vertex v of G )
        if ( color[v] == white )
            depthFirstFromVertex(G, v);
    return;
```

**Depth first search from a single vertex:** Performs depth-first search from a specific starting vertex.

```c
void depthFirstFromVertex(Digraph G, Vertex start)
    discoverVertex(start);
    while ( not empty(S) )
        if ( adjList[top(S)] is nonempty )
            adjacent = removeVertex(adjList[top(S)]);
            if ( color[adjacent] == white )
                discoverVertex(adjacent);
            else if ( color[adjacent] == gray )
                Process back edge (top(S),adjacent);
            else if ( d[adjacent] < d[top(S)] )
                Process cross edge (top(S),adjacent);
            else
                Process forward edge (top(S),adjacent);
        else
            finishTopVertex();
    return;
```
**Discovering of a new vertex**: Discover a new vertex \( x \), adjacent to \( \text{top}(S) \). The new vertex is pushed on the stack, colored gray, and its discover time is recorded. It undergoes any application-specific preorder processing.

```c
void discoverVertex( Vertex x )
    push( S, x );
    color[\text{top}(S)] = \text{gray};
    d[\text{top}(S)] = \text{++time};
    \text{Preorder process vertex } \text{top}(S);
    \text{return;}
```

**Exiting from the current vertex**: Exit a vertex after all edges out of the vertex have been explored. Any application-specific postorder processing of the vertex is performed. The vertex is colored black, and its finish time is recorded. It is then popped from the stack.

```c
void finishTopVertex( )
    \text{Postorder process vertex } \text{top}(S);
    f[\text{top}(S)] = \text{++time};
    color[\text{top}(S)] = \text{black};
    \text{pop}(S)
    \text{return;}
```

The running time of this algorithm is \( \Theta(n+e) \), apart from the time for application-specific processing of the vertices and edges.