DFS: theory

```
DFS (G)  // RT = O(|V| + |E|).

   For each vertex u in V(G) do
        u.Color ← White // unvisited
        u.Parent ← Nil


   // global vars: add more if needed
   time ← 0



   for each vertex u in V(G) do
        if u.Color = White then
            DFS_Visit (u)
```

```
/* Each node is visited exactly once
   O(1) work is assigned to each node

   Each edge is checked twice
   O(1) work is done each time */
```

```
DFS_Visit (u)   // u is visited by DFS
// Precondition: u is white

    u.Color ← Gray // Being processed
    u.dis ← ++ time // Discovery time of u



    for each v in Adj[u] do
         // O(1) work done inside this loop
         //  is assigned to edge (u,v)
         if v.Color = White then
              v.Parent ← u
              DFS_Visit (v)


    u.Color ← Black // Done processing u
    u.fin ← ++ time // Finish time of u
```

DFS: implementation template

```
DFS (Graph G)

     for(Vertex u: G)
          u.seen = false
          u.parent = null







     for(Vertex u: G)
          if (! u.seen)
              DFSVisit (u)
```

```
DFSVisit (Vertex u)

      u.seen = true




      for(Edge e: u.Adj)
            v = e.otherEnd(u)
            if (! v.seen)
                 v.parent = u
                 DFSVisit (v)
```

Connected components

```
int DFS (Graph G)  // G: undirected

   for(Vertex u: G)
       u.seen = false
       u.parent = null



   cno = 0

   for(Vertex u: G)
       if (! u.seen)
          DFSVisit (u, ++cno)


    // return no. of components in G
    Return cno
```

```
DFSVisit (u, cno)

    u.seen = true

    u.cno = cno

    for(Edge e: u.Adj)
        v = e.otherEnd(u)
        if (! v.seen)
           v.parent = u
           DFSVisit (v, cno)
```

DAG topological order

```
Stack DFSTop (Graph G) // G: DAG

   for(Vertex u: G)
       u.seen = false
       u.parent = null



   Create a new stack of vertices S

   for(Vertex u: G)
       if (! u.seen)
          DFSVisit (u, S)


   Return S
```

```
DFSVisit (u, S)

    u.seen = true


    for(Edge e: u.Adj)
       v = e.otherEnd(u)
       if (! v.seen)
          v.parent = u
          DFSVisit (v)



    S.push(u)
```