DFS FOR STRONGLY C.C.
DIJKSTRA’S ALGORITHM
**DFS**

visited = boolean array indexed by V initialized to F

```python
def explore(v):
    visited[v] = T
    for each neighbor w of v:
        if not visited[w]:
            explore(w)
```

```python
def DFS:
    for each v in V:
        if not visited[v]:
            explore(v)
```
LINEARIZATION USING DFS

visited = boolean array indexed by V initialized to F
L = empty list

def explore (v)
    visited [v] = T
    for each neighbor w of v:
        if not visited [w]:
            explore (w)
    L = [v] + L

def linearize
    for each v in V
        if not visited [v]: explore (v)
CONNECTED COMPONENTS USING DFS

visited = boolean array indexed by V initialized to F
ca = integer array indexed by V initialized to 0

def explore (v, c)
    visited [v] = T; ca [v] = c
    for each neighbor w of v:
        if not visited [w]:
            explore (w, c)

def cc
    c = 0
    for each v in V
        if not visited [v]:
            c++
            explore (v, c)
STRONGLY CONNECTED COMPONENTS USING DFS

\( G^R = G \) with all edges reversed

\( L = \) output of linearization algorithm on \( G^R \)

Run \( CC \) algorithm on \( G \), enumerating vertices as in \( L \)
Let $S, T$ be s.c.c. of $G$ with $> 1$ edges from $S$ to $T$

Then first vertex of $S$ in $L$ comes before the first vertex of $T$
Let $S,T$ be s.c.c. of $G^r$ with $>1$ edges from $S$ to $T$.
Then first vertex of $S$ in $L$ comes before the first vertex of $T$. 

\[ V \mid w \mid \rightarrow Z \rightarrow 1 \]

\[ L \quad G^r \]

\[ T \quad G \]

\[ Z \rightarrow V \rightarrow S \rightarrow G \]
Shortest path

\[ Q = 1 \ldots d \]

\[ \text{dist} \begin{bmatrix} 0 & 1 & 3 & 2 & 4 \\ s & a & b & c & d \end{bmatrix} \]

\[ v = 5 \]

\[ \text{prec} = \text{array indexed by vertices initialized to NIL} \]
\[ \text{dist} = \text{array indexed by vertices initialized to } \infty \]
\[ Q = \text{priority queue of vertices indexed by dist[·]} \]
\[ \text{dist}[s] = 0 \]

For each \( v \) \( Q \).insert(\( v \))

while \( Q \) is not empty

\[ v = Q \).deletemin(\) \]

for each \( w \) neighbor of \( v \):

if \( \text{dist}[w] > \text{dist}[v] + \ell(v, w) \):

\[ \text{dist}[w] = \text{dist}[v] + \ell(v, w) \]

\[ Q \).decreasekey(\( w \)) \]

\[ \text{prec}[w] = v \]
At the end of each iteration, the value of \( \text{dist}[v] \) is equal to the length of the shortest path from \( s \) to \( v \) that uses only nodes outside \( Q \) as intermediate steps, and it is correct \( s \rightarrow v \) distance if \( v \) is outside \( Q \).

- **First iteration**
  
  \[
  \begin{align*}
  d[s] &= 0 \\
  d[v] &= d(s,v) \text{ if } v \text{ neighbor of } s \\
  d[v] &= \infty \text{ for others} \\
  Q \text{ contains all vertices except } s
  \end{align*}
  \]

- **Suppose this is true after \( t \) iterations**

  - Consider iteration \( t+1 \)

  - V removed from \( Q \) at time \( t+1 \)

  - \( v \) not in \( Q \) at time \( t \)

  - in \( Q \) at time \( t \)