DFS FOR STRONGLY C.C.

DIJKSTRA'S ALGORITHM

# DFS

visited = boolean array indexed by V initialized to F

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


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

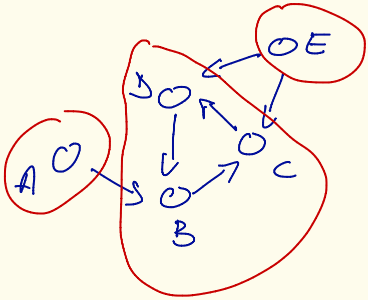
```
def explore(v, c)
    visited[v] = T ; ca[v] = c
    for each neighbor w of v:
        if not visited[w]:
            explore(w, c)


def cc
    c = 0
    for each v in V
        if not visited[v]:
            c++
            explore(v, c)
```

# STRONGLY CONNECTED COMPONENTS USING DFS

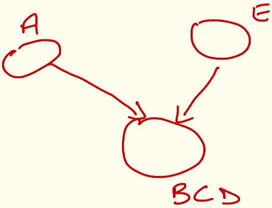
$G^R = G$ with all edges reversed

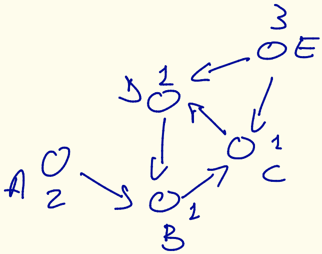
$L$ = output of linearization algorithm on $G^R$

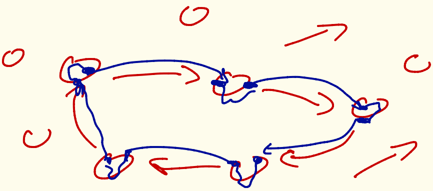
Run CC algorithm on $G$, enumerating vertices as in $L$




A , E , BCD



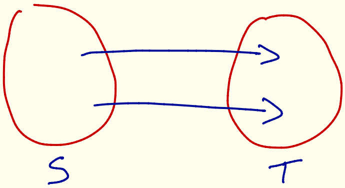



G directed graph

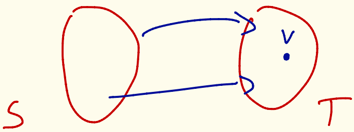
L output of LINEARIZE alg on G

(list of vertices in reverse order of termination of explore(v))




Let S, T be s.c.c. of G with $\geq 1$ edges from S to T

Then first vertex of S in L comes before the first vertex of T




First vertex to be discovered in $S \cup T$ is $v \in S$

Then all of S, all of T is discovered inside of explore(v)

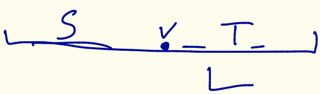
explore(v) terminates after all of T

v comes before all of T in L

First vertex to be discovered in $S \cup T$

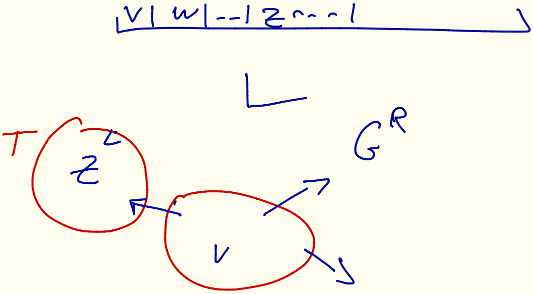
$v \in T$

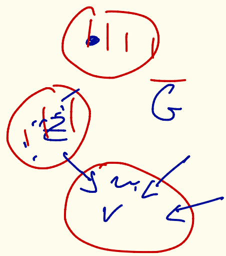
explore(v) terminates discovering all of T but at at time in which no vertex of is visited



Let $S, T$ be s.c.c. of $G^R$ with $\geq 1$ edges from $S$ to $T$

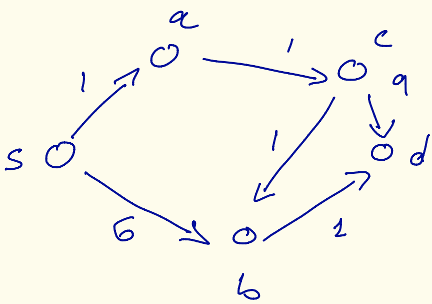
Then first vertex of $S$ in $L$ comes before the first vertex of $T$



# Shortest path

$Q = \{ \; . \quad . \quad , \quad d \}$




$$\text{dist} \; \begin{array}{ccccc} 0 & 1 & 3 & 2 & 4 \\ s & a & b & c & d \end{array}$$

$v = b$

prec = array indexed by vertices initialized to NIL
dist = array indexd by vertices initialized to $\infty$
Q = priority queue of vertices indexed by dist[·]

```
dist[s] = 0
 For each v  Q.insert(v)
 while Q is not empty
     v = Q.deletemin()
     for each w neighbor of v :
         if  dist[w] > dist[v] + ℓ(v,w) :
             dist[w] = dist[v] + ℓ(v,w)
               Q.decreasekey(w)
               prec[w] = v
```

At the end of each iteration the value of dist[v] is equal to length of shortest path from s to v that uses only nodes outside Q as intermediate steps, and it is correct s->v distance if v is outside Q

- First iteration

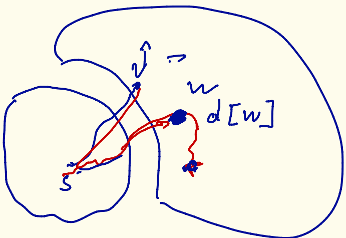
  $d[s] = 0$

  $d[v] = \ell(s,v)$ if v neighbor of s

  $d[v] = \infty$ for others

  Q contains all vertices except s

- Suppose this is true after t iterations
  consider iteration t+1




v removed from Q at time t+1

not in Q at time t

in Q at time t