

# SEARCHING IN GRAPHS

We already did the most basic form of search:  
(neighbor query)

# SEARCHING IN GRAPHS

We already did the most basic form of search:  
(neighbor query)

In general, given a vertex  $s$   
we want to locate some vertex  $t$ ,  
↓  
find a path in  $G$

# SEARCHING IN GRAPHS

We already did the most basic form of search:  
(neighbor query)

In general, given a vertex  $s$   
we want to locate some vertex  $t$ ,  
↓  
find a path in  $G$

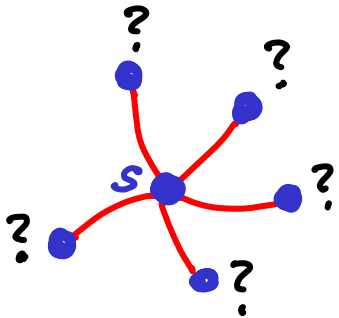
or we want to visit all vertices,  
in a "local" organized manner.

# BREADTH FIRST SEARCH (BFS)

# BREADTH FIRST SEARCH (BFS)

Start by checking if  $t$  is a neighbor of  $s$ .

↳ look one step away from  $s$ .

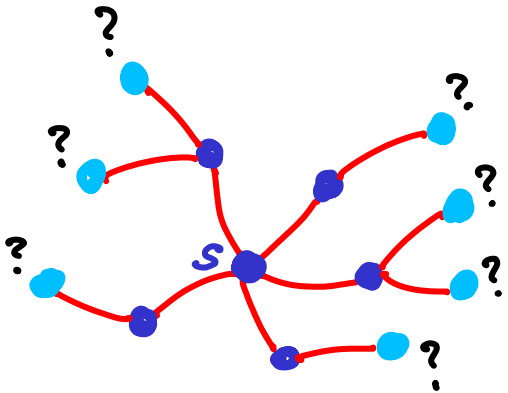


# BREADTH FIRST SEARCH (BFS)

Start by checking if  $t$  is a neighbor of  $s$ .

↳ look one step away from  $s$ .

If yes, done. If not, then check all neighbors-of-neighbors



# BREADTH FIRST SEARCH (BFS)

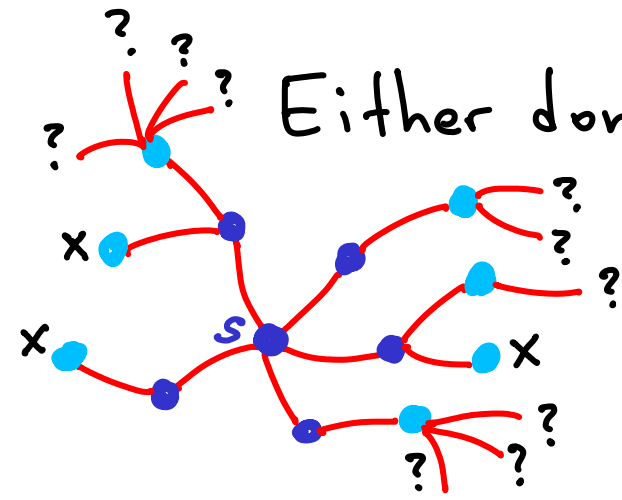
Start by checking if  $t$  is a neighbor of  $s$ .

↳ look one step away from  $s$ .

If yes, done. If not, then check all neighbors-of-neighbors

↳ one step from each.

Either done, or repeat (dig deeper)



# BREADTH FIRST SEARCH (BFS)

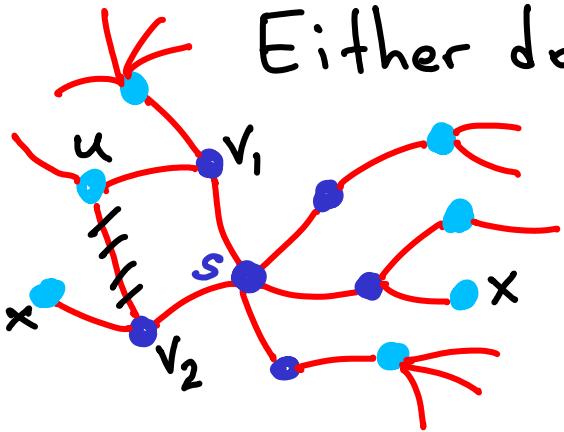
Start by checking if  $t$  is a neighbor of  $s$ .

↳ look one step away from  $s$ .

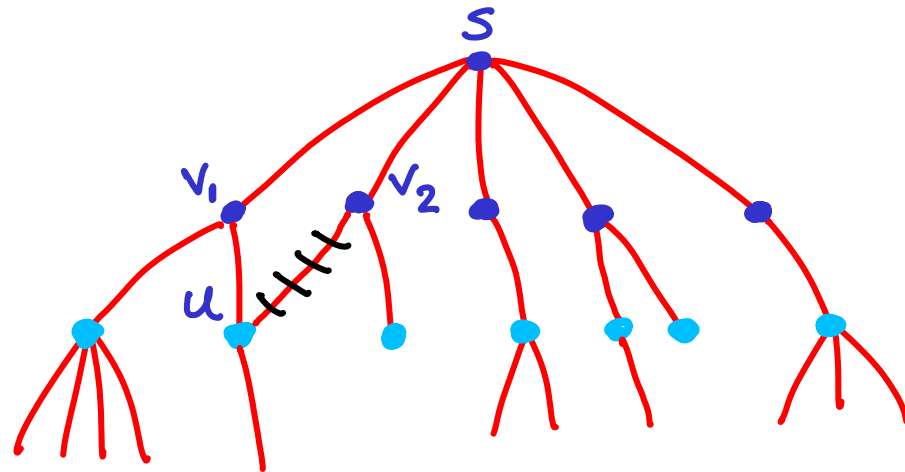
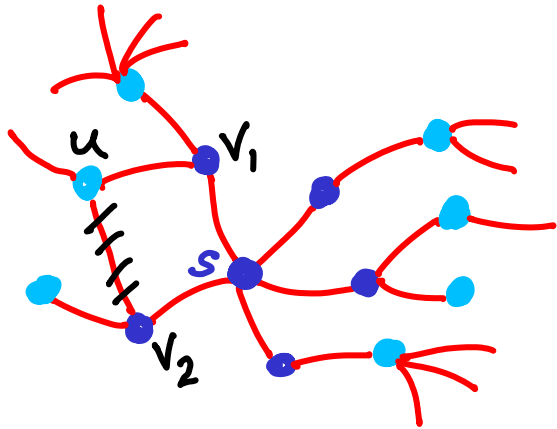
If yes, done. If not, then check **all** <sup>unexplored!</sup> neighbors-of-neighbors

↳ one step from each.

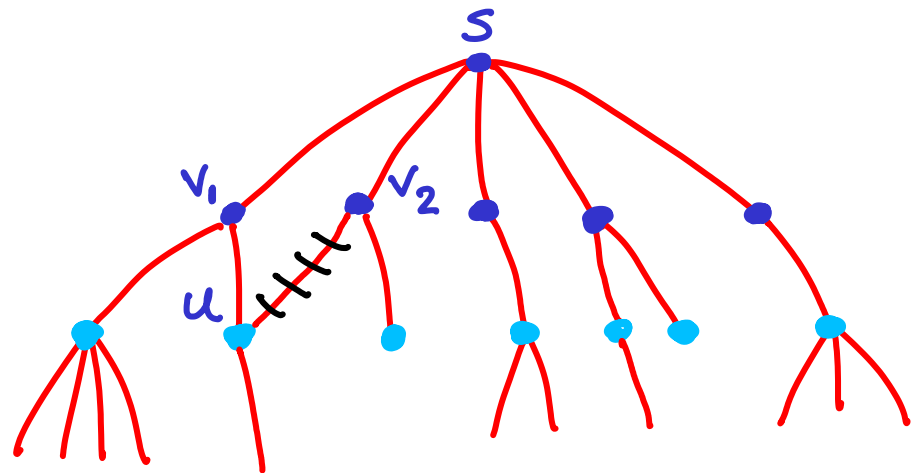
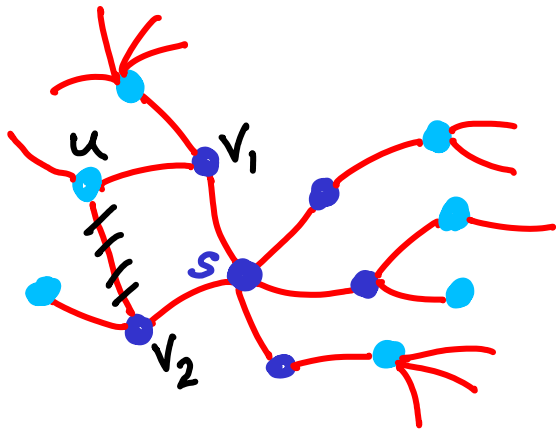
Either done, or repeat (dig deeper) ... only on unexplored neighbors!





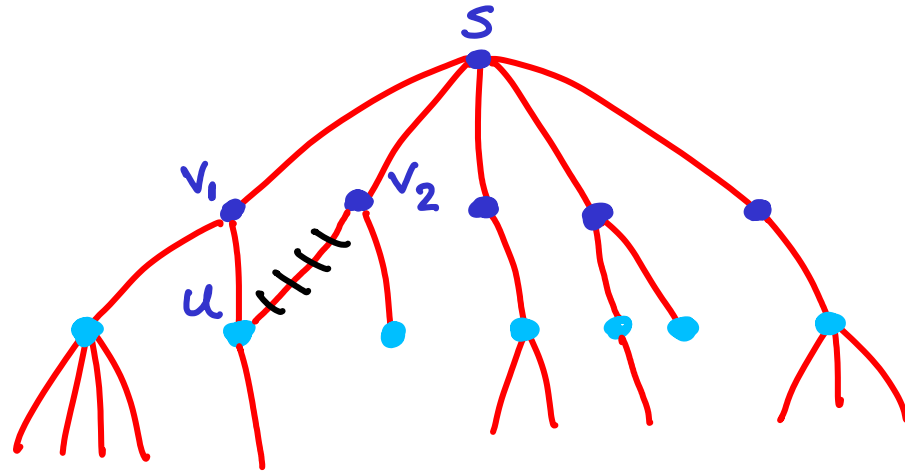
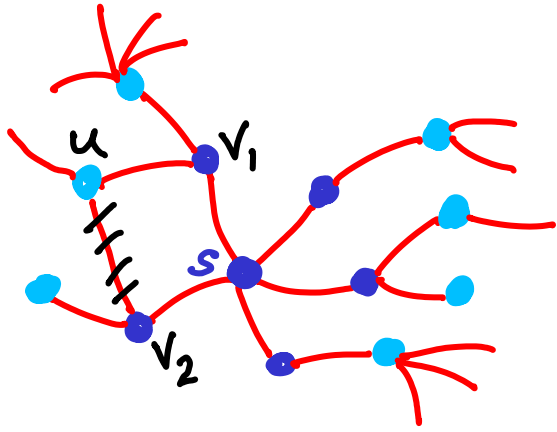


Search follows a tree pattern.



Search follows a tree pattern.

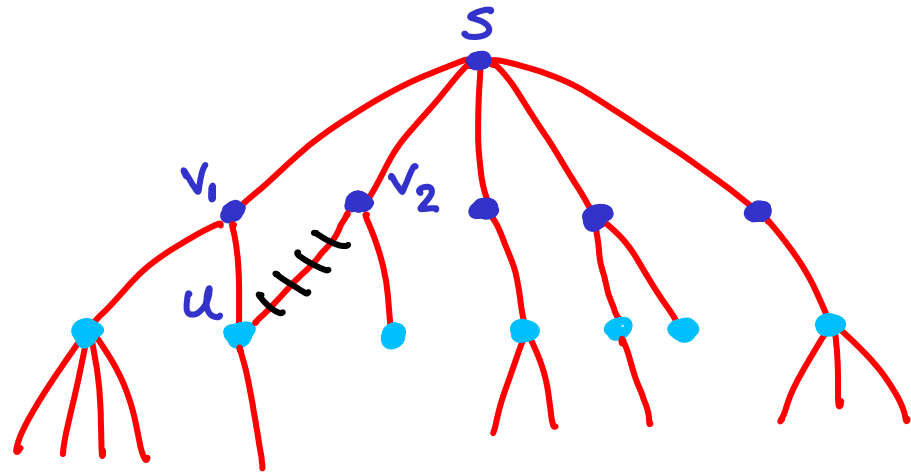
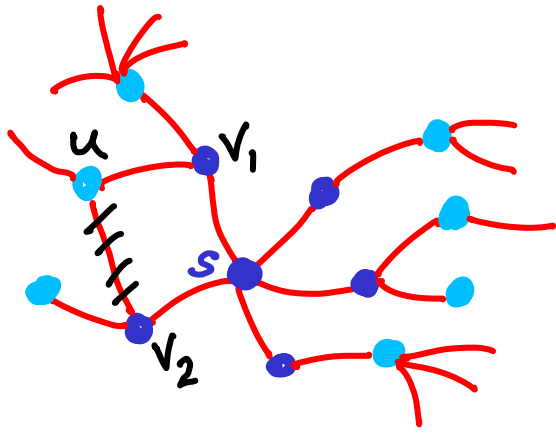
BFS extends depth by 1 at all possible nodes



Search follows a tree pattern.

BFS extends depth by 1 at all possible nodes

↳ always processing nodes closer to  $s$  first

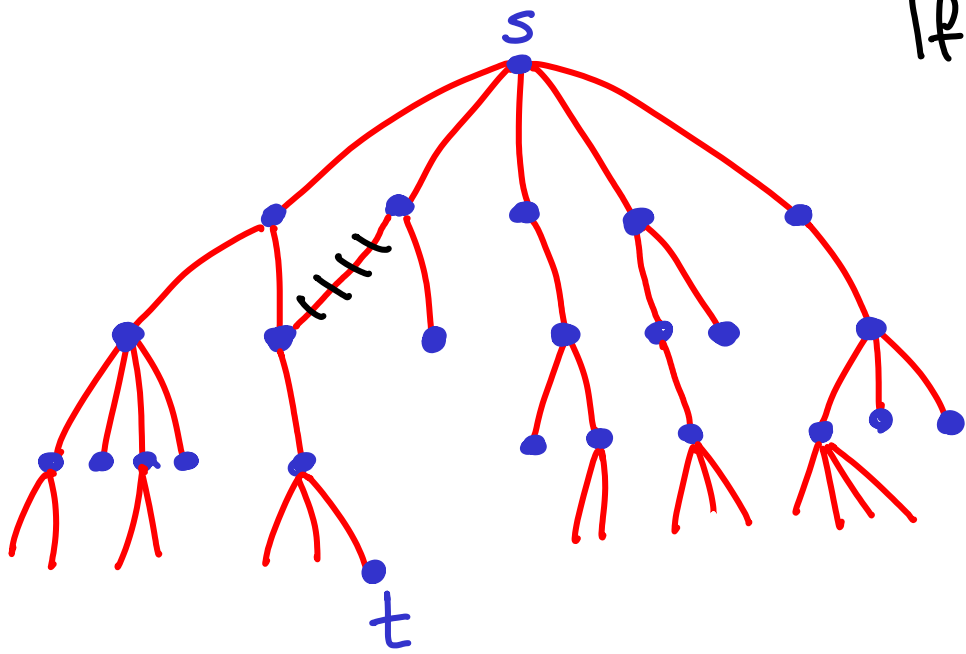


Search follows a tree pattern.

BFS extends depth by 1 at all possible nodes

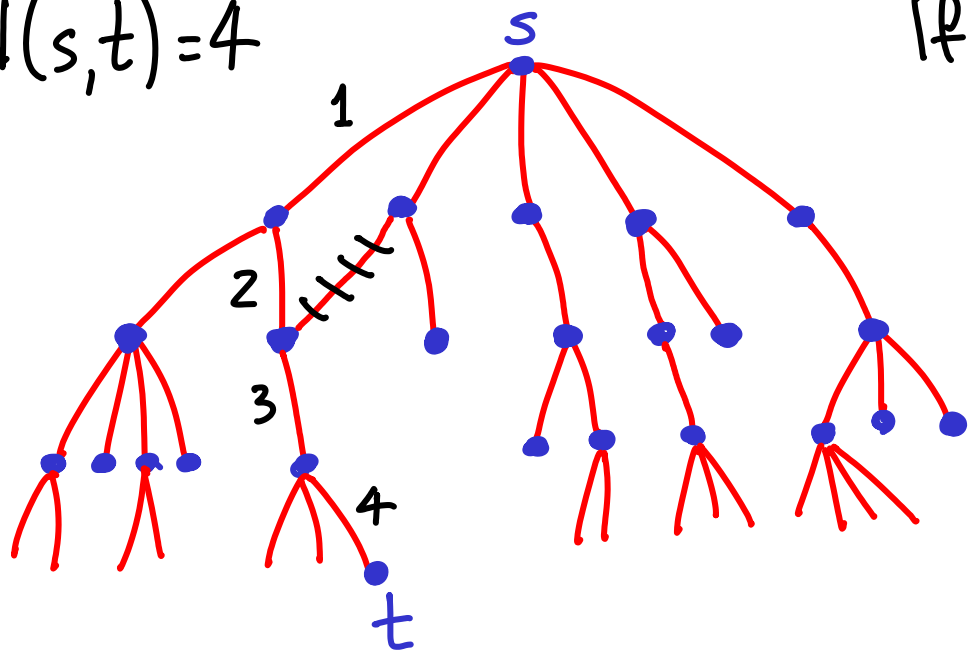
↳ always processing nodes closer to  $s$  first

↳ if any node is rediscovered, pretend it didn't happen (eg.  $u$ )



If  $s$  &  $t$  are in the same  
connected component  
then the search will find  $t$ .

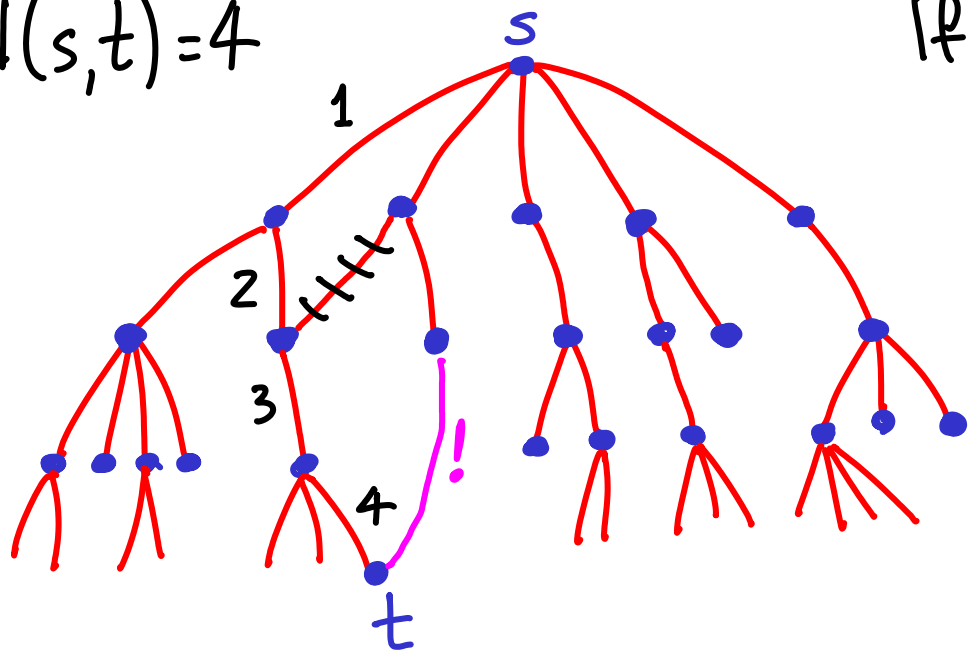
$$d(s, t) = 4$$



If  $s$  &  $t$  are in the same connected component then the search will find  $t$ .

Even better, BFS will find a shortest path  $s \rightarrow t$ .

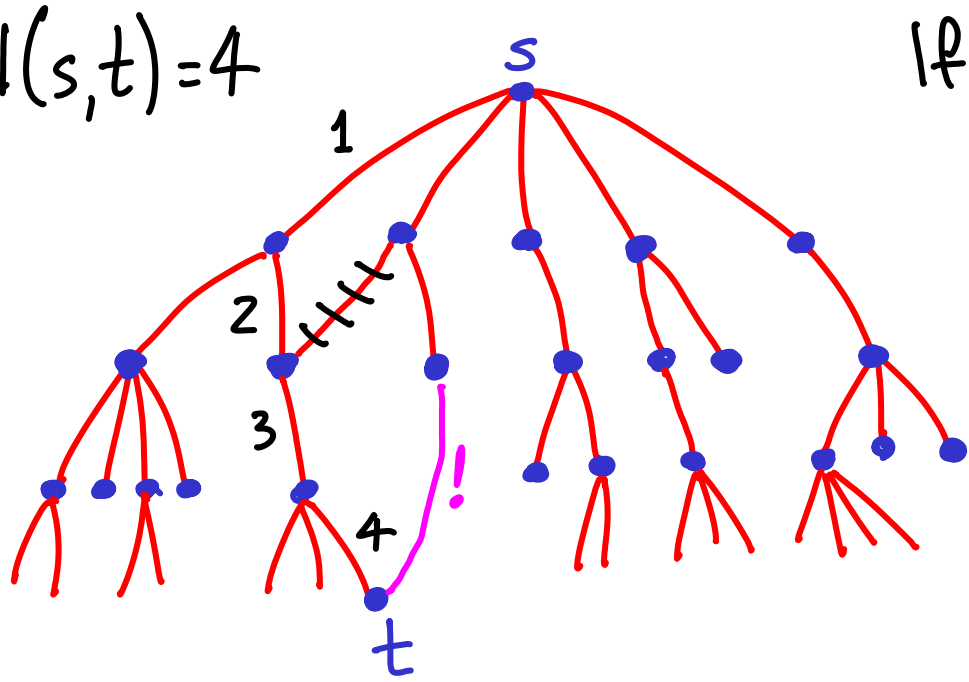
$$d(s,t)=4$$



If  $s$  &  $t$  are in the same connected component then the search will find  $t$ .

Even better, BFS will find a shortest path  $s \rightarrow t$ .  
(prove by contradiction)

$$d(s,t)=4$$



If  $s$  &  $t$  are in the same connected component then the search will find  $t$ .

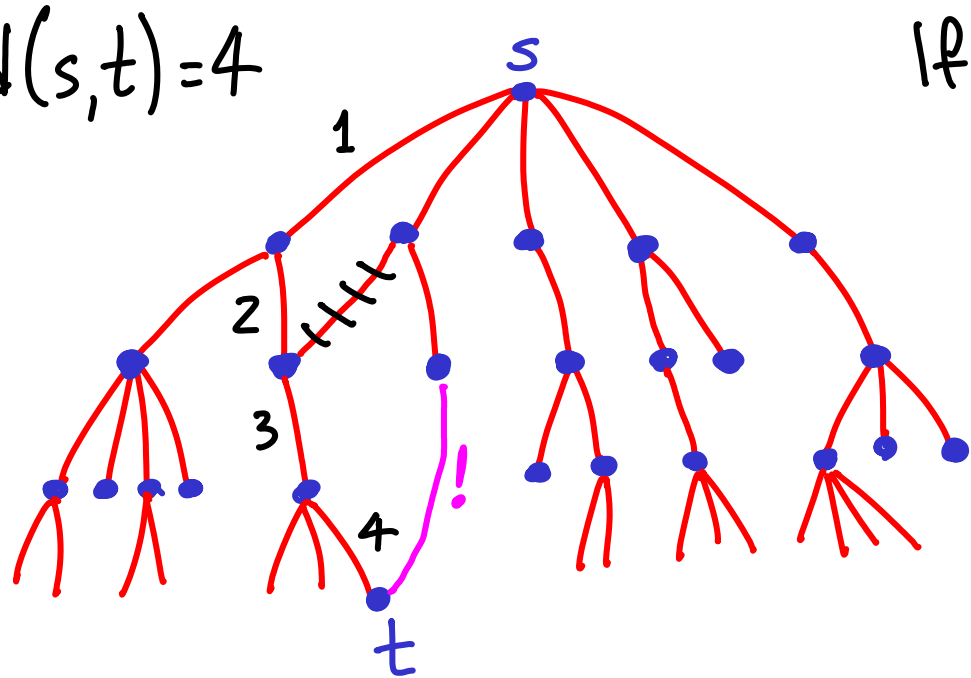
Even better, BFS will find a shortest path  $s \rightarrow t$ .

(prove by contradiction)

time? (supposing we can tell instantly whether a vertex is "new")



$$d(s, t) = 4$$



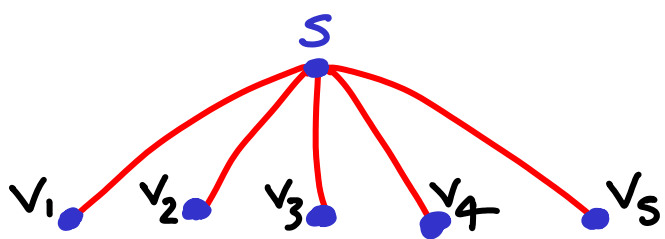
If  $s$  &  $t$  are in the same connected component then the search will find  $t$ .

Even better, BFS will find a shortest path  $s \rightarrow t$ .

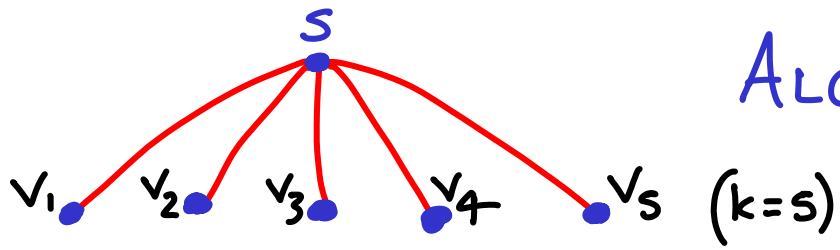
(prove by contradiction)

time? (supposing we can tell instantly whether a vertex is "new")

$O(|E|)$  (in component of  $s$ )



ALGO: (0) mark  $s$   
(i) check  $Adj[s] : v_1, \dots, v_k$   
 $(k=s)$  ↪ if  $v_i = t$  DONE



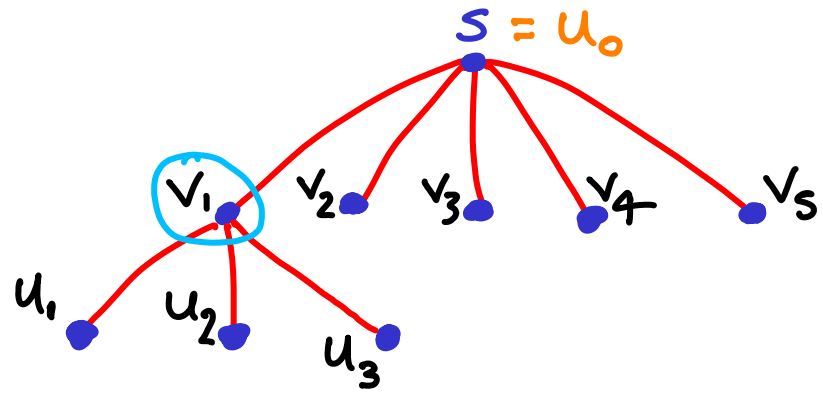
ALGO: (0) mark  $s$

(1) check  $Adj[s] : v_1 \dots v_k$

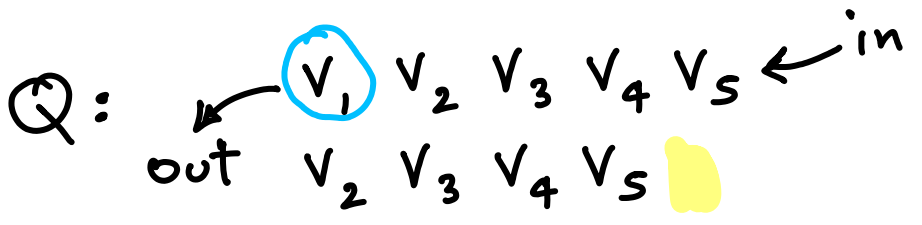
if  $v_i = t$  DONE  
if  $v_i \neq t$  • mark as visited  
• put in queue : Q

Q:  $v_1$   $v_2$   $v_3$   $v_4$   $v_s$  ← in  
out

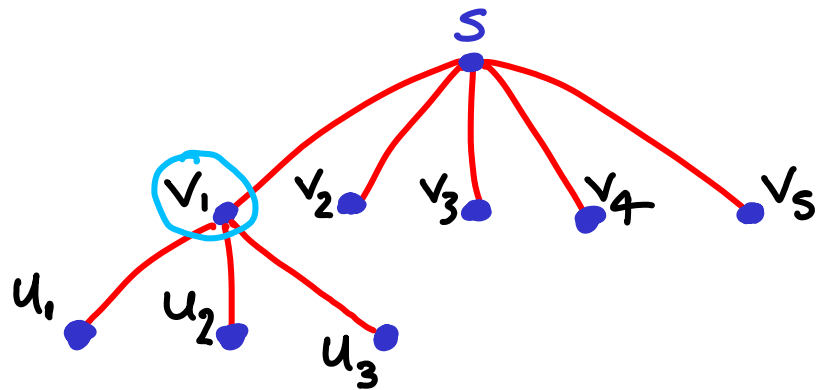
Why a queue?



ALGO: (0) mark  $s$   
 (1) check  $Adj[s] : v_1 \dots v_k$   
 if  $v_i = t$  DONE  
 if  $v_i \neq t$  • mark as visited  
 • put in queue :  $Q$



(2) While  $Q$  not empty,  
 - remove first vertex  $v_f$  in  $Q$   
 - check  $Adj[v_f] : u_0 \dots u_3$   
 ↑  
 ?



ALGO: (0) mark  $s$

(1) check  $Adj[s] : v_1 \dots v_k$

↪ if  $v_i = t$  DONE  
    • mark as visited  
 ↪ if  $v_i \neq t$  • put in queue :  $Q$

(2) While  $Q$  not empty,

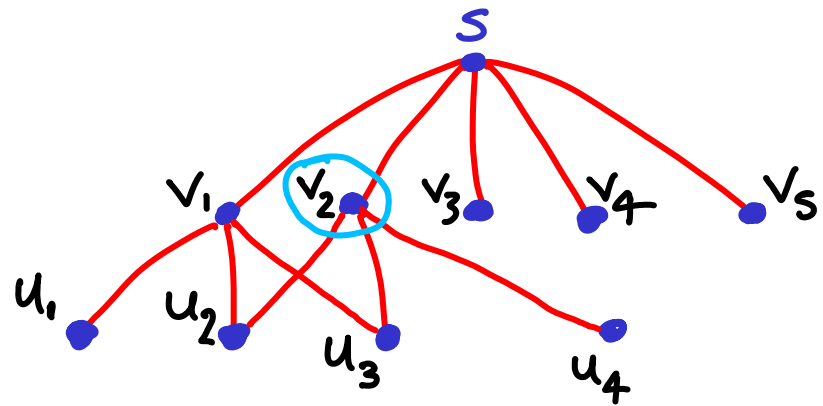
- remove first vertex  $v_f$  in  $Q$

- check  $Adj[v_f] : u_0 \dots u_3$

↪ if  $u_i = t$  DONE

↪ if  $u_i \neq t$  & unmarked  
    put  $u_i$  in  $Q$ .  
    mark  $u_i$

$Q$ :    ↖     $v_1$   $v_2$   $v_3$   $v_4$   $v_5$     ↖ in  
       out  $v_2$   $v_3$   $v_4$   $v_5$      $u_1$   $u_2$   $u_3$

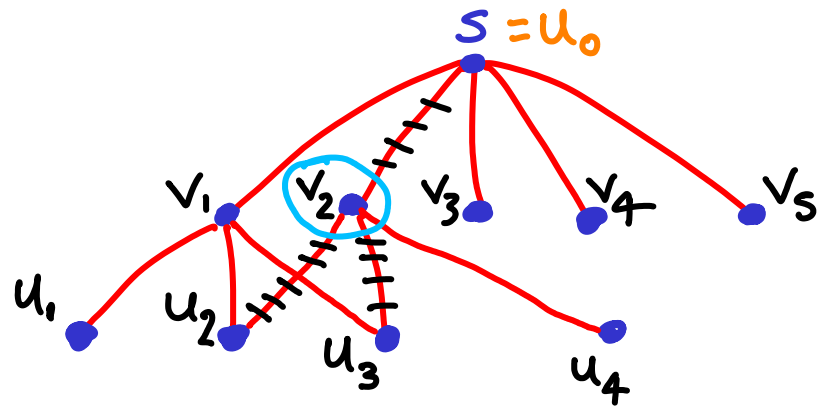


ALGO: (0) mark  $s$

(1) check  $Adj[s] : v_1 \dots v_k$   
 ↳ if  $v_i = t$  DONE  
 ↳ if  $v_i \neq t$  • mark as visited  
 • put in queue = Q

Q:  $v_1, v_2, v_3, v_4, v_5 \leftarrow in$   
 out  $v_2, v_3, v_4, v_5, u_1, u_2, u_3$   
 $v_3, v_4, v_5, u_1, u_2, u_3, u_4$

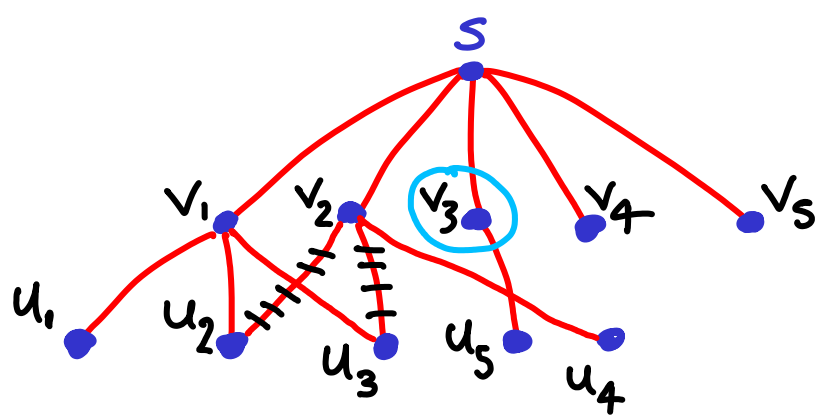
(2) While Q not empty,  
 - remove first vertex  $v_f$  in Q  
 - check  $Adj[v_f]$   
 ↳ if  $u_i = t$  DONE  
 ↳ if  $u_i \neq t$  & unmarked  
 put  $u_i$  in Q.  
 mark  $u_i$



ALGO: (0) mark  $s$   
 (1) check  $Adj[s] : v_1, \dots, v_k$   
 ↳ if  $v_i = t$  DONE  
   • mark as visited  
 ↳ if  $v_i \neq t$  • put in queue :  $Q$

$Q$ :  
 out  $v_1$   $v_2$   $v_3$   $v_4$   $v_5$   $u_1$   $u_2$   $u_3$  ← in  
 $v_2$   $v_3$   $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   
 $v_3$   $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   $u_4$

(2) While  $Q$  not empty,  
 - remove first vertex  $v_f$  in  $Q$   
 - check  $Adj[v_f] : u_0, u_2, u_3, u_4$   
 ↳ if  $u_i = t$  DONE  
 ↳ if  $u_i \neq t$  & unmarked  
   put  $u_i$  in  $Q$ .  
   mark  $u_i$ .



ALGO: (0) mark  $s$

(1) check  $Adj[s] : v_1, \dots, v_k$

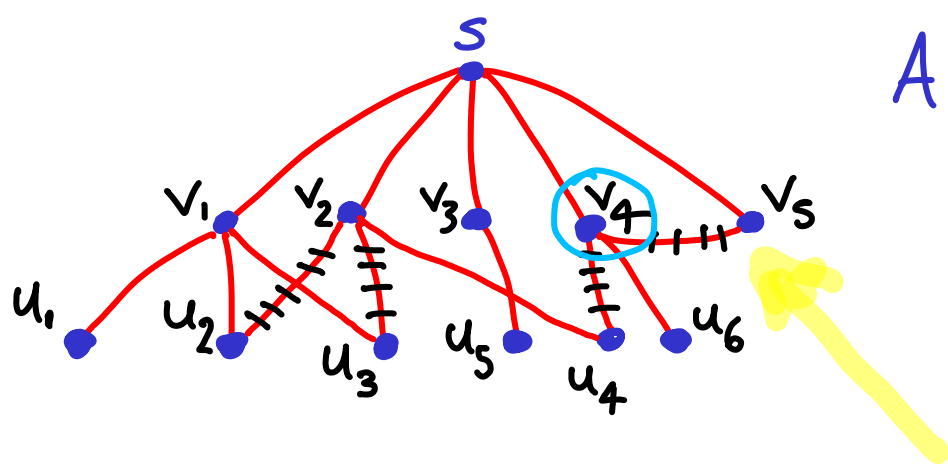
↪ if  $v_i = t$  DONE  
     • mark as visited  
 ↪ if  $v_i \neq t$  • put in queue :  $Q$

$Q :$   
 out     $v_1$   $v_2$   $v_3$   $v_4$   $v_5$  ← in  
        $v_2$   $v_3$   $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   
        $v_3$   $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   $u_4$   
        $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   $u_4$   $u_5$

(2) While  $Q$  not empty,

↪ - remove first vertex  $v_f$  in  $Q$   
    ↪ - check  $Adj[v_f] : u_0, u_s$   
       ↪ if  $u_i = t$  DONE  
       ↪ if  $u_i \neq t$  & unmarked  
           put  $u_i$  in  $Q$ .  
           mark  $u_i$



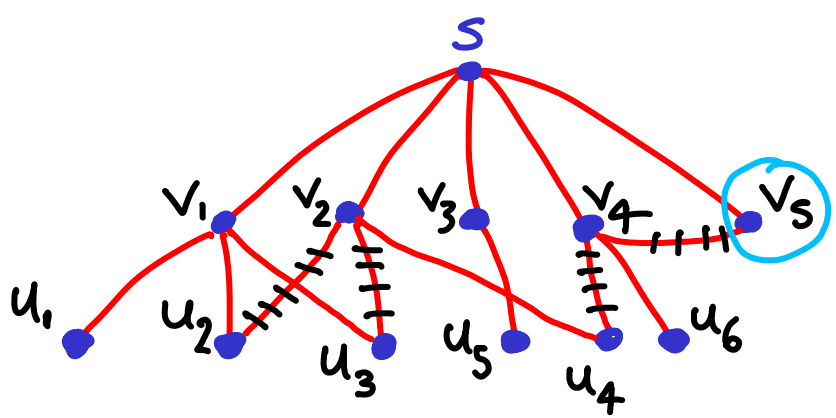


ALGO: (0) mark  $s$

(1) check  $Adj[s] : v_1 \dots v_k$   
 ↳ if  $v_i = t$  DONE  
 ↳ if  $v_i \neq t$  • mark as visited  
 • put in queue :  $Q$

$Q$ :  
 out  $v_1$   $v_2$   $v_3$   $v_4$   $v_5$  ← in  
 $v_2$   $v_3$   $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   
 $v_3$   $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   $u_4$   
 $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   $u_4$   $u_5$   
 $v_5$   $u_1$   $u_2$   $u_3$   $u_4$   $u_5$   $u_6$

(2) While  $Q$  not empty,  
 - remove first vertex  $v_f$  in  $Q$   
 - check  $Adj[v_f] : u_1 \dots u_p$   
 ↳ if  $u_i = t$  DONE  
 ↳ if  $u_i \neq t$  & unmarked  
 put  $u_i$  in  $Q$ .  
 mark  $u_i$



ALGO: (0) mark  $s$

(1) check  $Adj[s] : v_1 \dots v_k$

↪ if  $v_i = t$  DONE  
    • mark as visited  
 ↪ if  $v_i \neq t$  • put in queue :  $Q$

(2) While  $Q$  not empty,

- remove first vertex  $v_f$  in  $Q$   
 - check  $Adj[v_f] : u_1 \dots u_p$

↪ if  $u_i = t$  DONE  
 ↪ if  $u_i \neq t$  & unmarked  
    put  $u_i$  in  $Q$ .  
    mark  $u_i$

$Q$ :  
 out     $v_1$   $v_2$   $v_3$   $v_4$   $v_5$  ← in  
        $v_2$   $v_3$   $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   
        $v_3$   $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   $u_4$   
        $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   $u_4$   $u_5$   
        $v_5$   $u_1$   $u_2$   $u_3$   $u_4$   $u_5$   $u_6$   
        $u_1$   $u_2$   $u_3$   $u_4$   $u_5$   $u_6$

ALGO: (0) mark  $s$

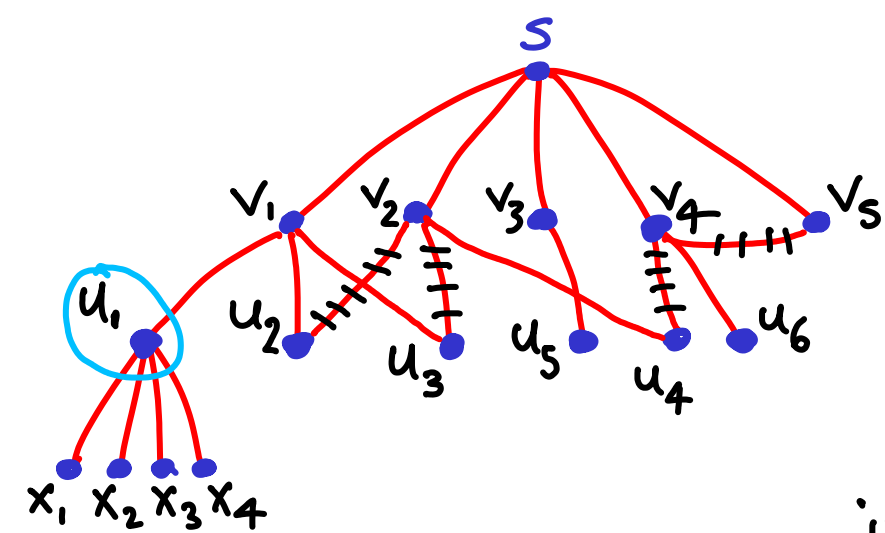
(i) check  $Adj[s] : v_1 \dots v_k$

↪ if  $v_i = t$  DONE  
    • mark as visited  
 ↪ if  $v_i \neq t$  • put in queue :  $Q$

(2) While  $Q$  not empty,

- remove first vertex  $v_f$  in  $Q$   
 - check  $Adj[v_f] : u_1 \dots u_p$

↪ if  $u_i = t$  DONE  
 ↪ if  $u_i \neq t$  & unmarked  
    put  $u_i$  in  $Q$ .  
    mark  $u_i$



$Q$ :    out     $v_1$   $v_2$   $v_3$   $v_4$   $v_5$  ← in  
            $v_2$   $v_3$   $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   
            $v_3$   $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   $u_4$   
            $v_4$   $v_5$   $u_1$   $u_2$   $u_3$   $u_4$   $u_5$   
            $v_5$   $u_1$   $u_2$   $u_3$   $u_4$   $u_5$   $u_6$   
            $u_1$   $u_2$   $u_3$   $u_4$   $u_5$   $u_6$   
            $u_2$   $u_3$   $u_4$   $u_5$   $u_6$   $x_1$   $x_2$   $x_3$   $x_4$

ALGO: (0) mark  $s$

(1) check  $Adj[s] : v_1, \dots, v_k$

↪ if  $v_i = t$  DONE  
    • mark as visited  
 ↪ if  $v_i \neq t$  • put in queue :  $Q$

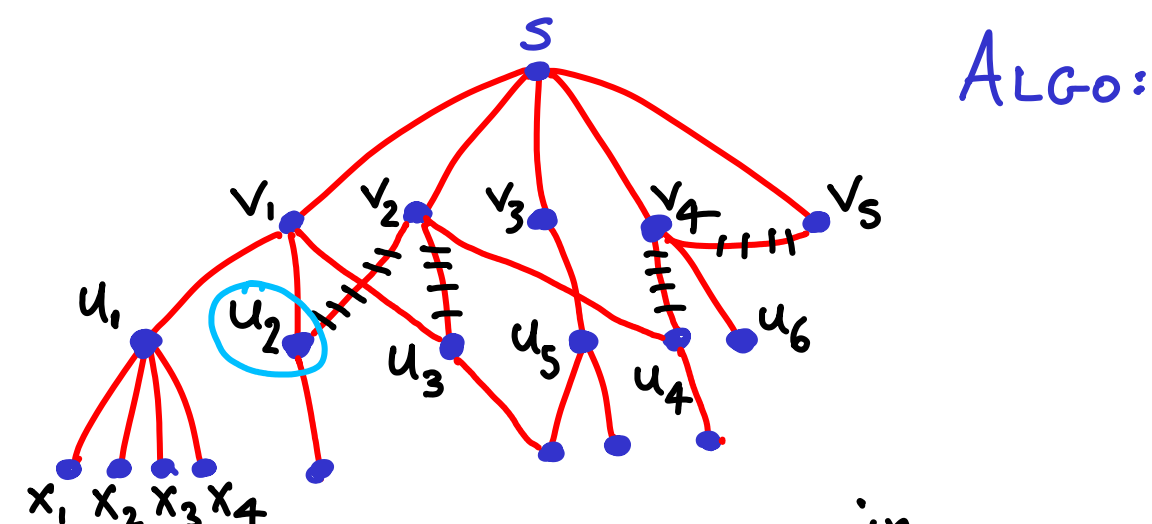
(2) While  $Q$  not empty,

- remove first vertex  $v_f$  in  $Q$

- check  $Adj[v_f] : u_1, \dots, u_p$

↪ if  $u_i = t$  DONE

↪ if  $u_i \neq t$  & unmarked  
    put  $u_i$  in  $Q$ .  
    mark  $u_i$



$Q$ :     $v_1, v_2, v_3, v_4, v_5$  ← in  
       out  $v_2, v_3, v_4, v_5, \underline{u_1}, \underline{u_2}, \underline{u_3}$   
            $v_3, v_4, v_5, u_1, \underline{u_2}, \underline{u_3}, \underline{u_4}$   
            $v_4, v_5, u_1, u_2, u_3, u_4, \underline{u_5}$   
            $v_5, u_1, u_2, u_3, \underline{u_4}, u_5, \underline{u_6}$   
            $u_1, u_2, u_3, u_4, u_5, u_6$   
            $u_2, u_3, u_4, u_5, u_6, \underline{x_1}, \underline{x_2}, \underline{x_3}, \underline{x_4}$     etc

(0) mark  $s$

(1) check  $\text{Adj}[s] : v_1, \dots, v_k$

↳ if  $v_i = t$  DONE

↳ if  $v_i \neq t$  • mark as visited  
• put in queue :  $Q$

(2) While  $Q$  not empty,

- remove first vertex  $v_f$  in  $Q$

- check  $\text{Adj}[v_f] : u_1, \dots, u_p$

↳ if  $u_i = t$  DONE

↳ if  $u_i \neq t$  & unmarked

put  $u_i$  in  $Q$ .  
mark  $u_i$

- mark  $s$  & put in  $Q$ .
- $\text{depth}(s) = 0$

- While  $Q$  not empty,

$x = \text{dequeue}(Q)$

check  $\text{Adj}[x] : u_1, \dots, u_p :$

↳ if  $u_i$  is unmarked

mark  $u_i$  & put in  $Q$ .

$\text{depth}(u_i) = 1 + \text{depth}(x)$

$\text{parent}(u_i) = x ; u_i \rightarrow \text{child}(x)$

just the search process

(0) mark  $s$

(1) check  $\text{Adj}[s] : v_1, \dots, v_k$

↳ if  $v_i = t$  DONE

↳ if  $v_i \neq t$  • mark as visited  
• put in queue :  $Q$

(2) While  $Q$  not empty,

- remove first vertex  $v_f$  in  $Q$

- check  $\text{Adj}[v_f] : u_1, \dots, u_p$

↳ if  $u_i = t$  DONE

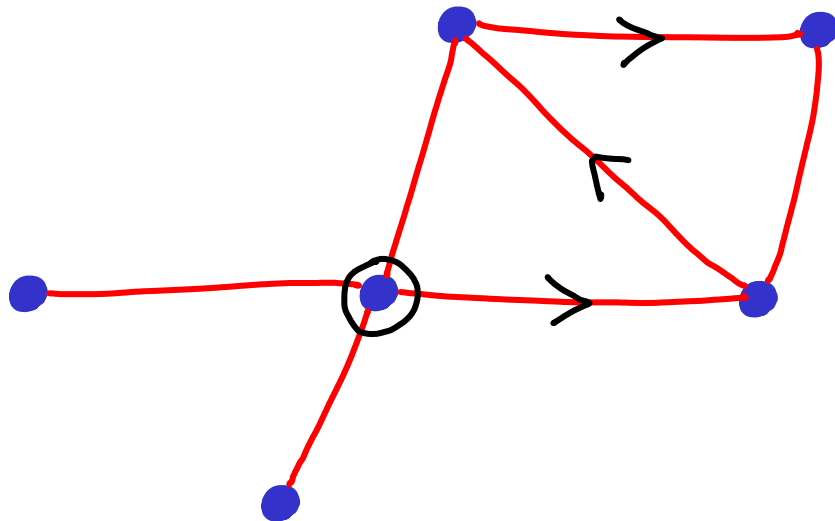
↳ if  $u_i \neq t$  & unmarked

put  $u_i$  in  $Q$ .  
mark  $u_i$

# DEPTH FIRST SEARCH (DFS)

# DEPTH FIRST SEARCH (DFS)

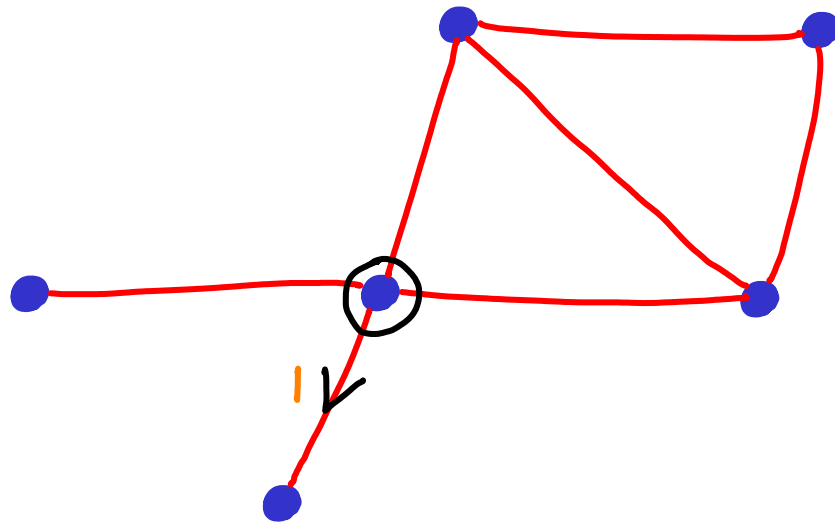
- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.





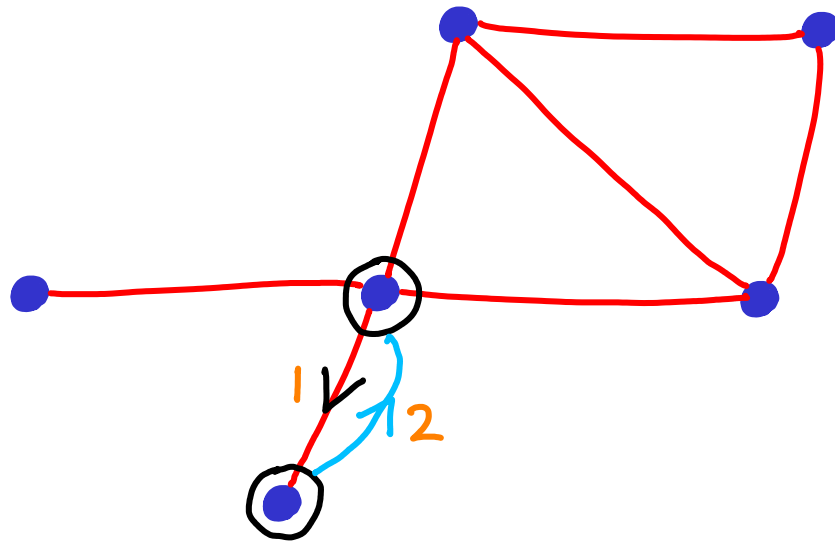
# DEPTH FIRST SEARCH (DFS)

- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.



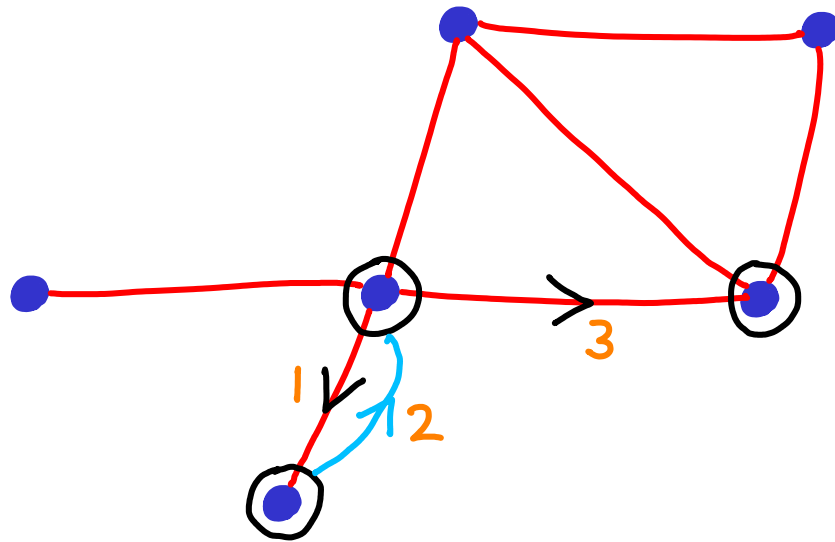
# DEPTH FIRST SEARCH (DFS)

- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.



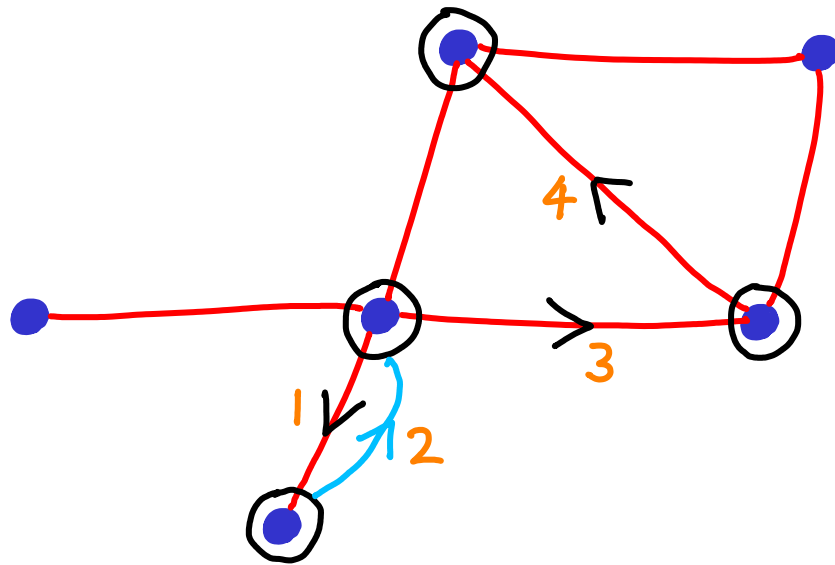
# DEPTH FIRST SEARCH (DFS)

- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.



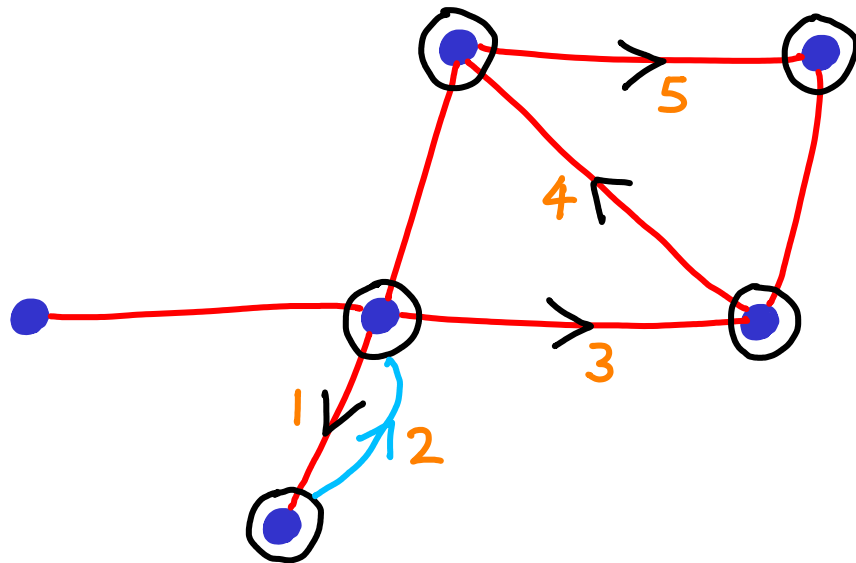
# DEPTH FIRST SEARCH (DFS)

- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.



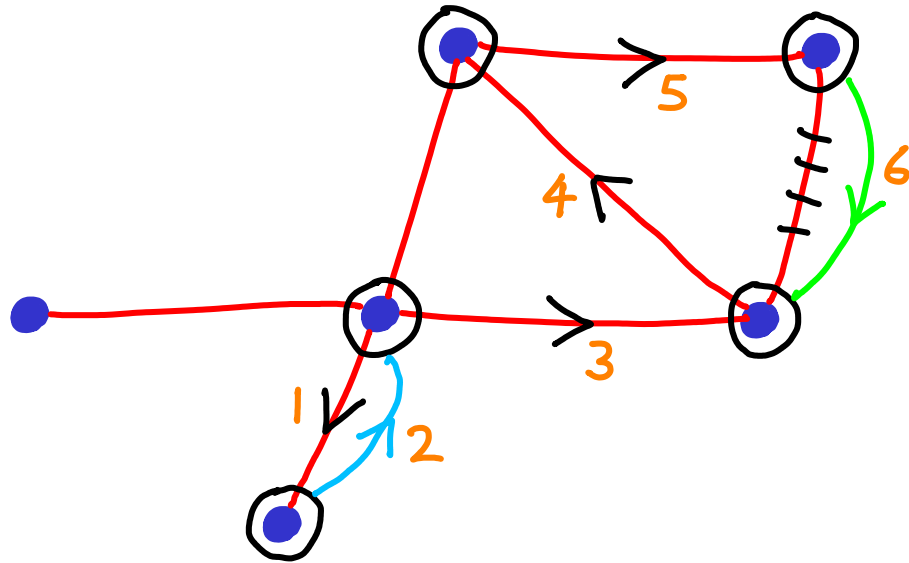
# DEPTH FIRST SEARCH (DFS)

- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.



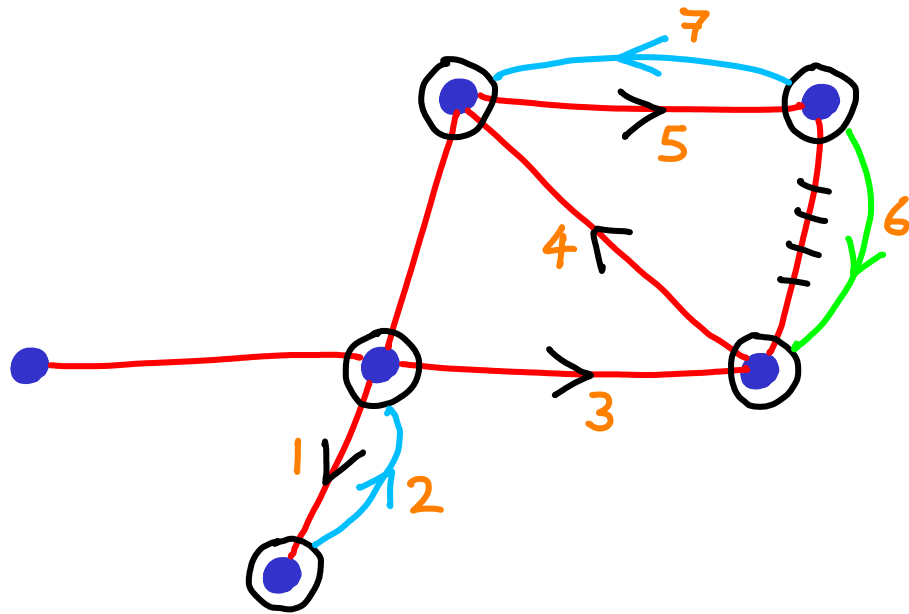
# DEPTH FIRST SEARCH (DFS)

- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.



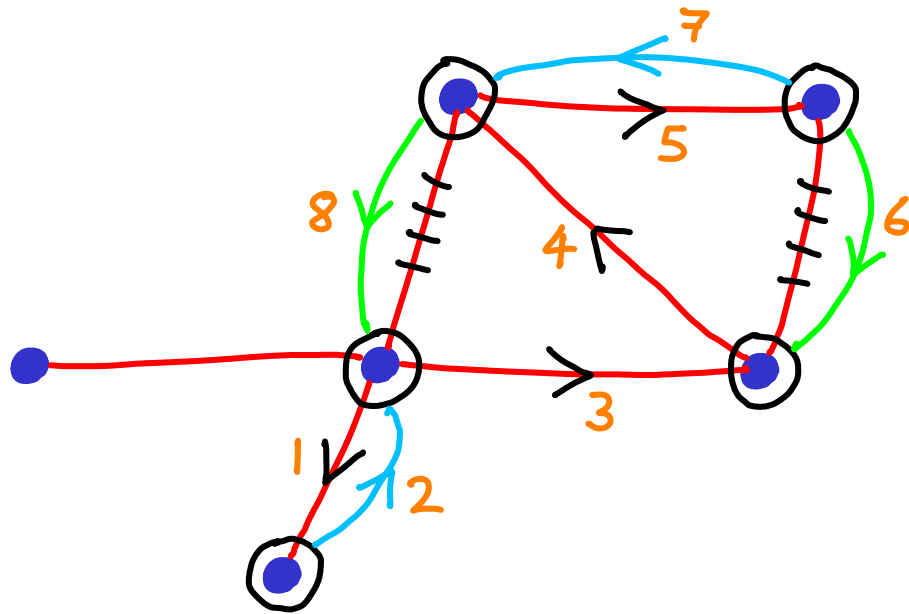
# DEPTH FIRST SEARCH (DFS)

- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.



# DEPTH FIRST SEARCH (DFS)

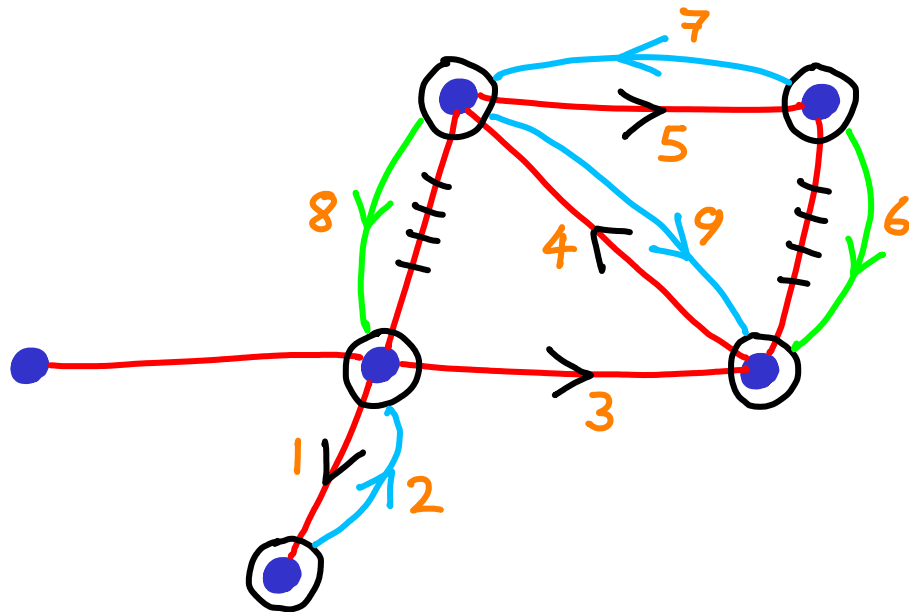
- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.





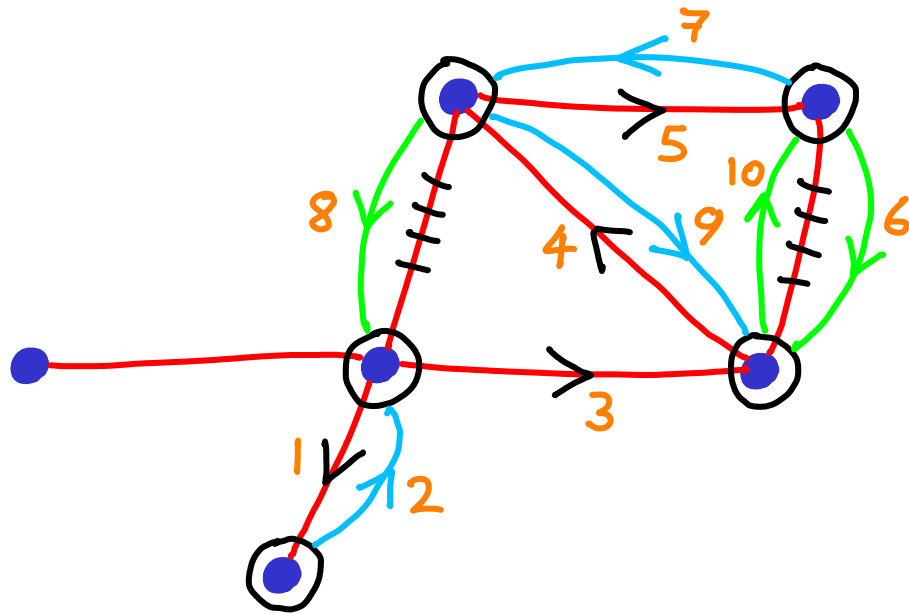
# DEPTH FIRST SEARCH (DFS)

- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.



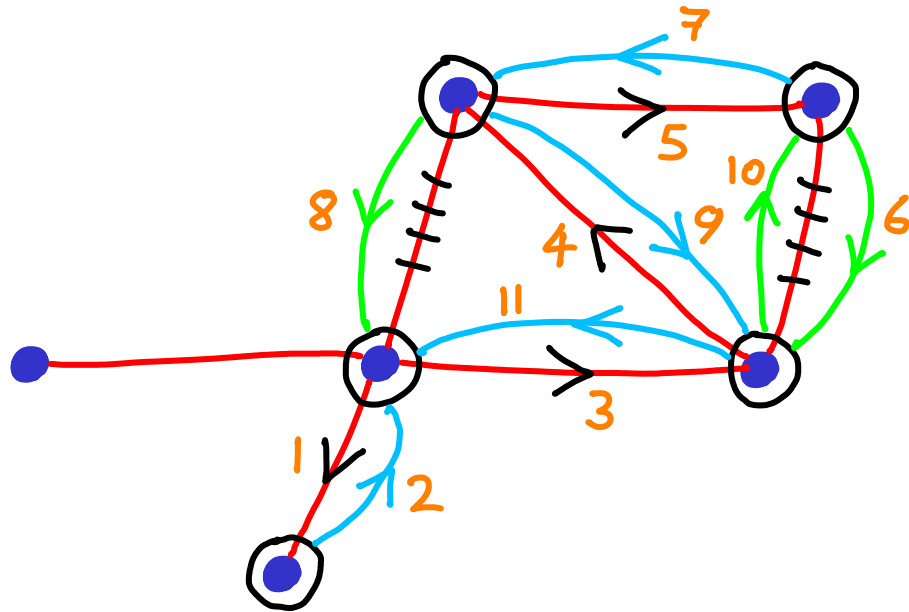
# DEPTH FIRST SEARCH (DFS)

- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.



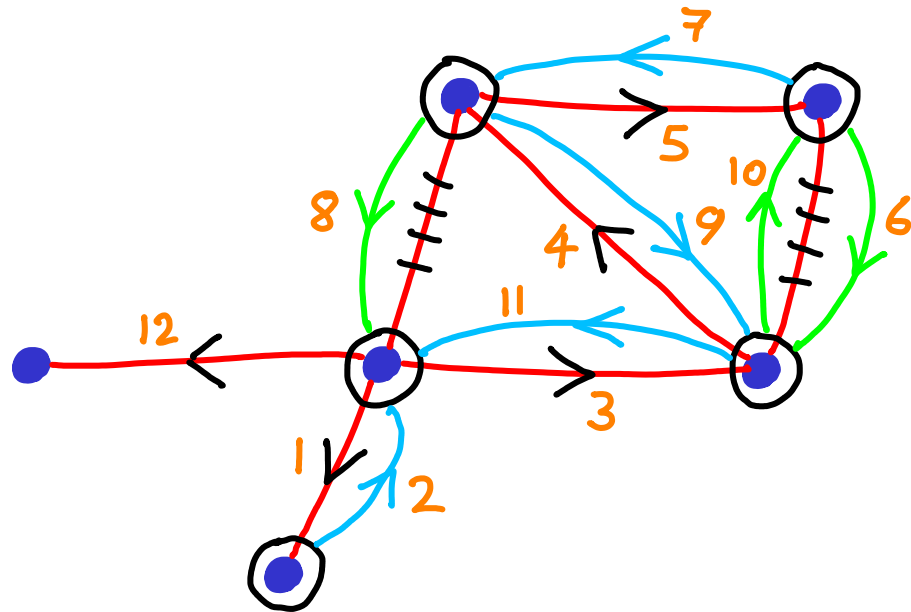
# DEPTH FIRST SEARCH (DFS)

- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.



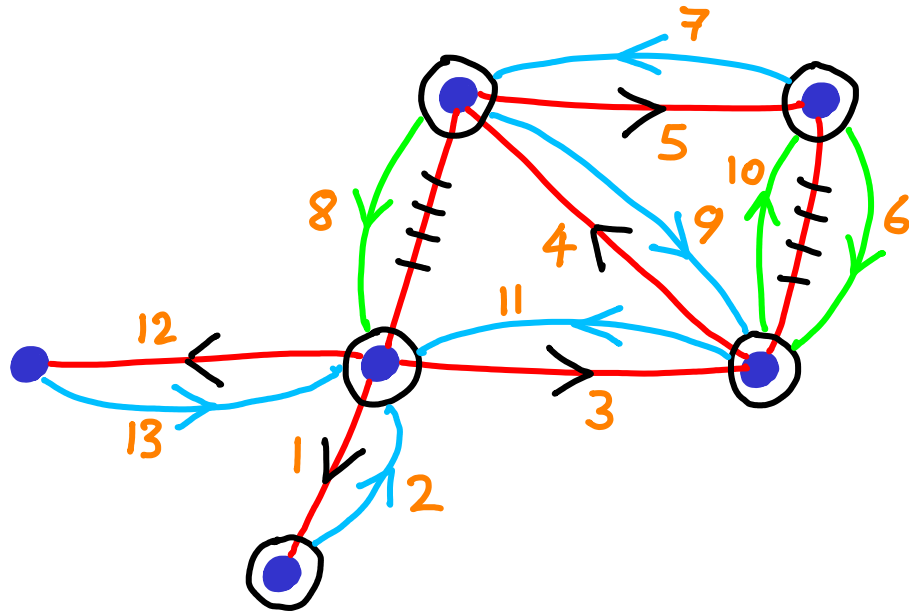
# DEPTH FIRST SEARCH (DFS)

- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.



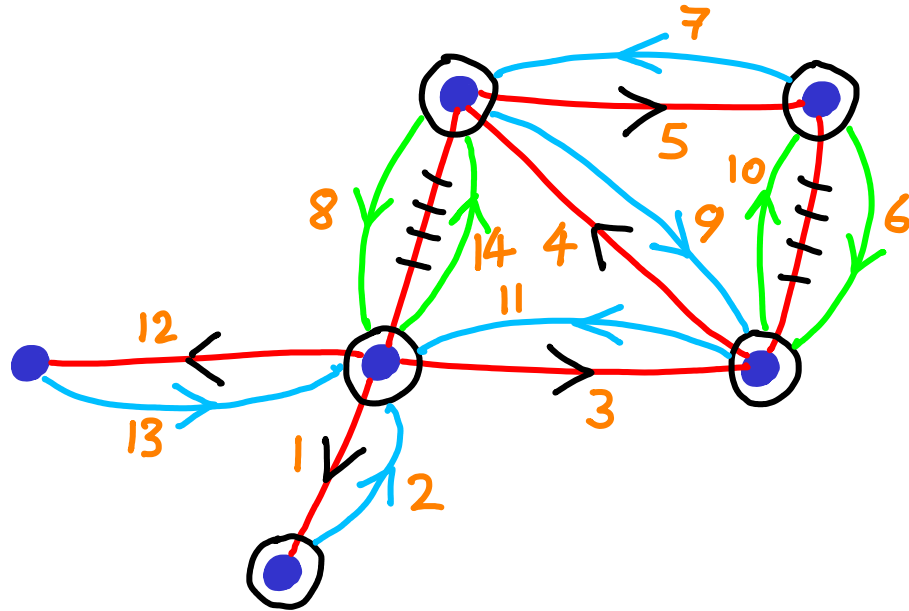
# DEPTH FIRST SEARCH (DFS)

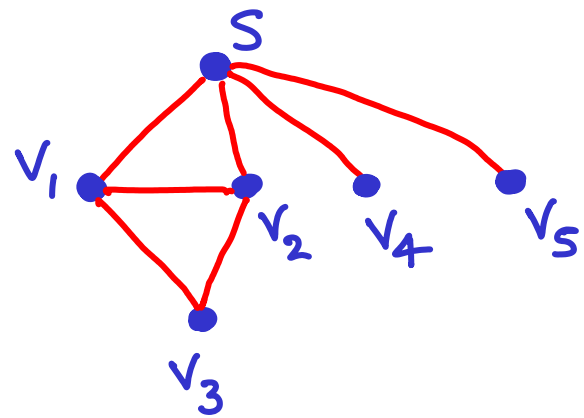
- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.

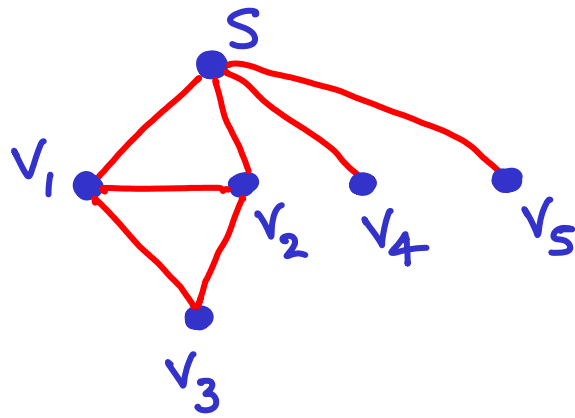


# DEPTH FIRST SEARCH (DFS)

- Follow an unvisited path for as long as possible.
- When you reach a vertex w/ only previously-visited neighbors, back up (from where you came from) & try again.







Adjacency list

$S \rightarrow v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_5$

$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow S$

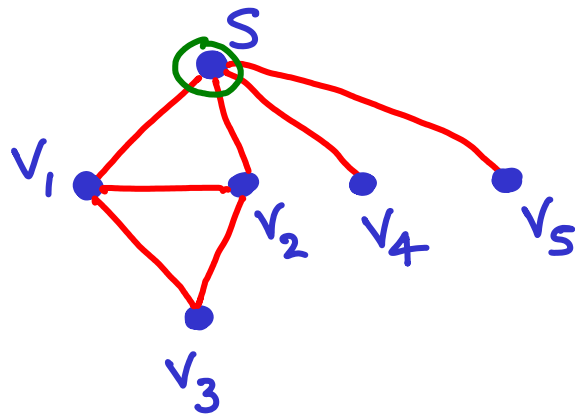
$v_2 \rightarrow S \rightarrow v_3 \rightarrow v_1$

$v_3 \rightarrow v_2 \rightarrow v_1$

$v_4 \rightarrow S$

$v_5 \rightarrow S$





As with BFS, mark visited nodes.

Adjacency list

$S \rightarrow v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_5$

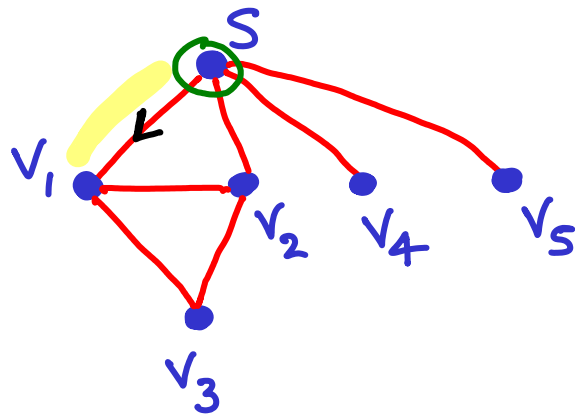
$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow S$

$v_2 \rightarrow S \rightarrow v_3 \rightarrow v_1$

$v_3 \rightarrow v_2 \rightarrow v_1$

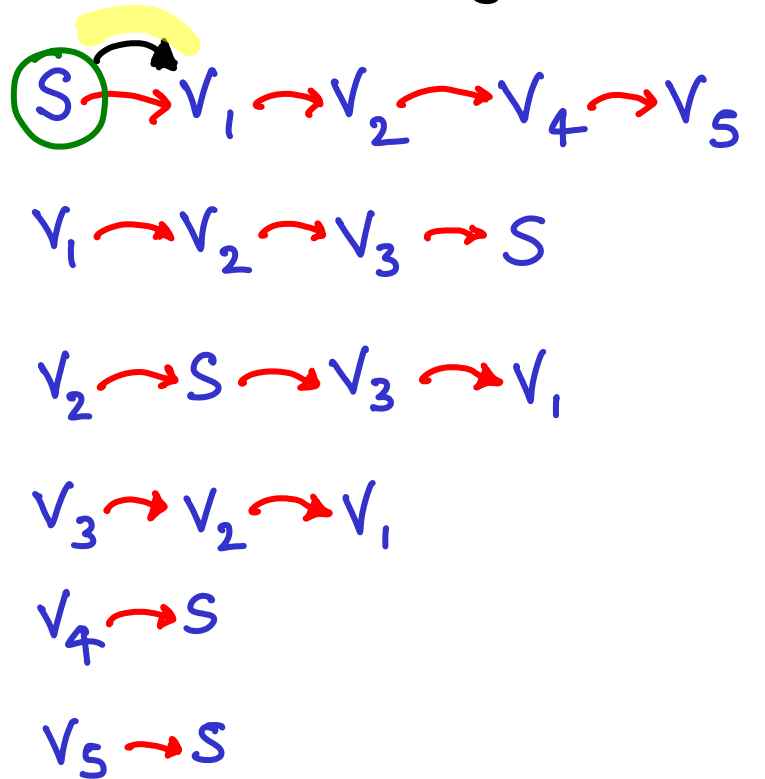
$v_4 \rightarrow S$

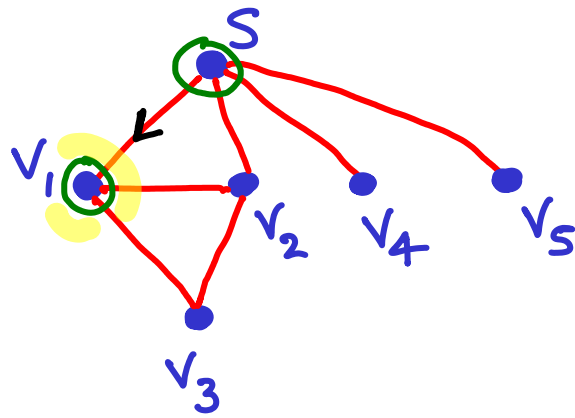
$v_5 \rightarrow S$



As with BFS, mark visited nodes.

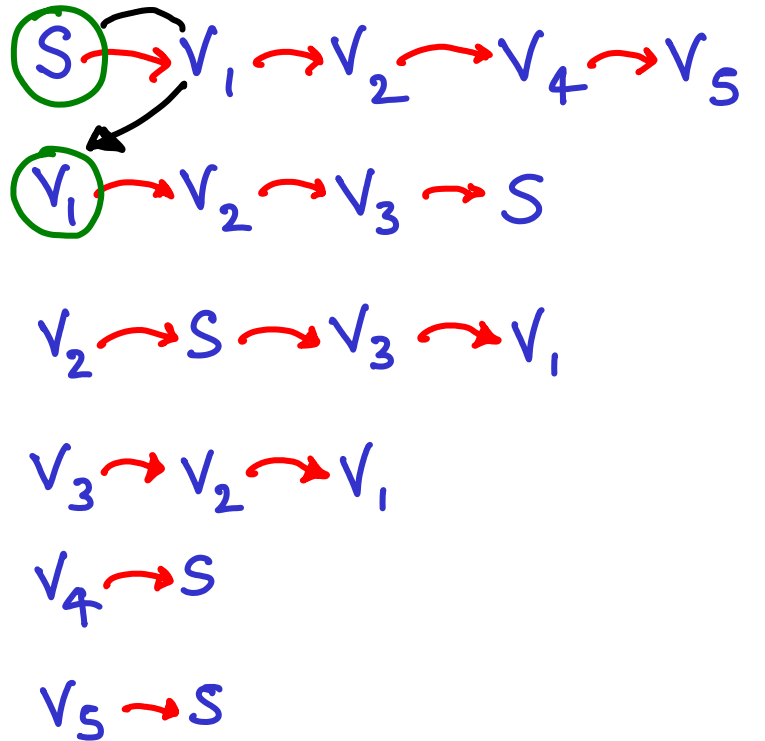
Adjacency list

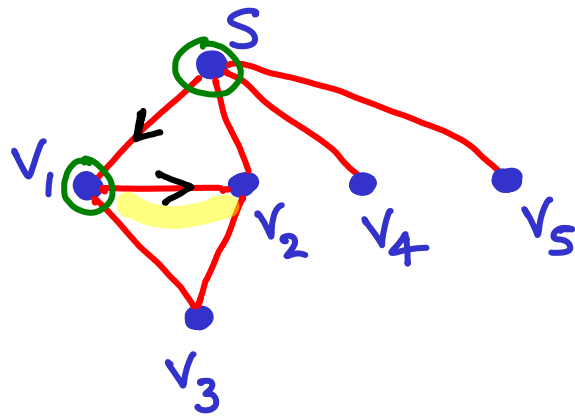




As with BFS, mark visited nodes.

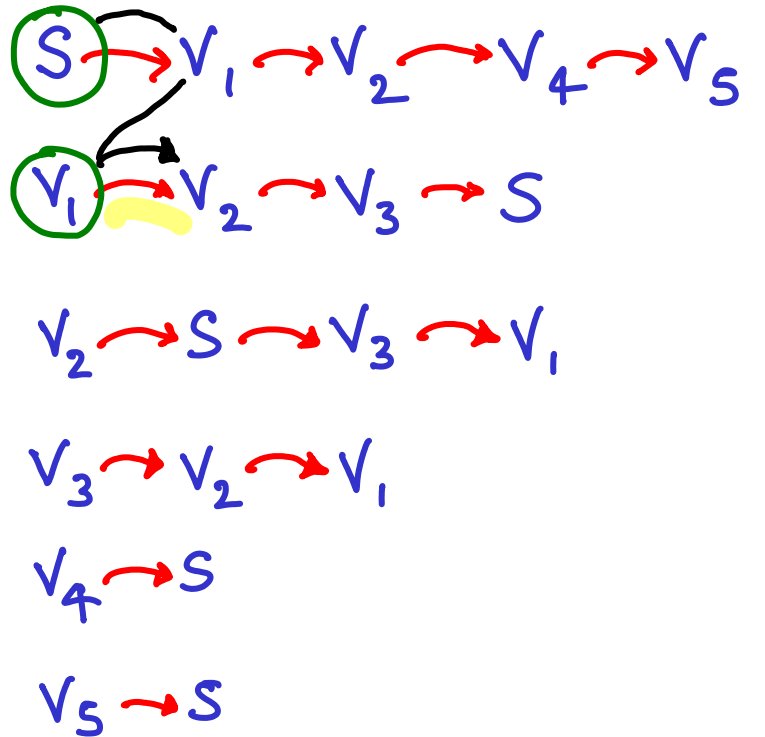
Adjacency list

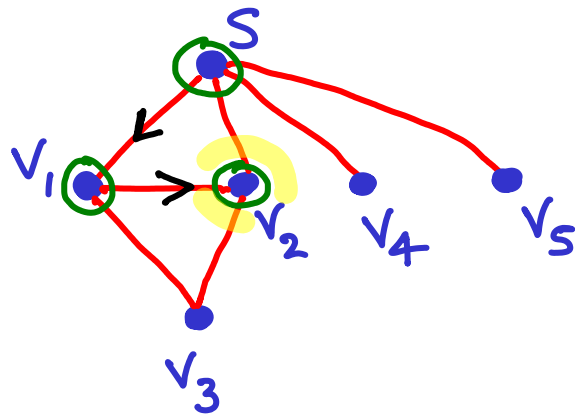




As with BFS, mark visited nodes.

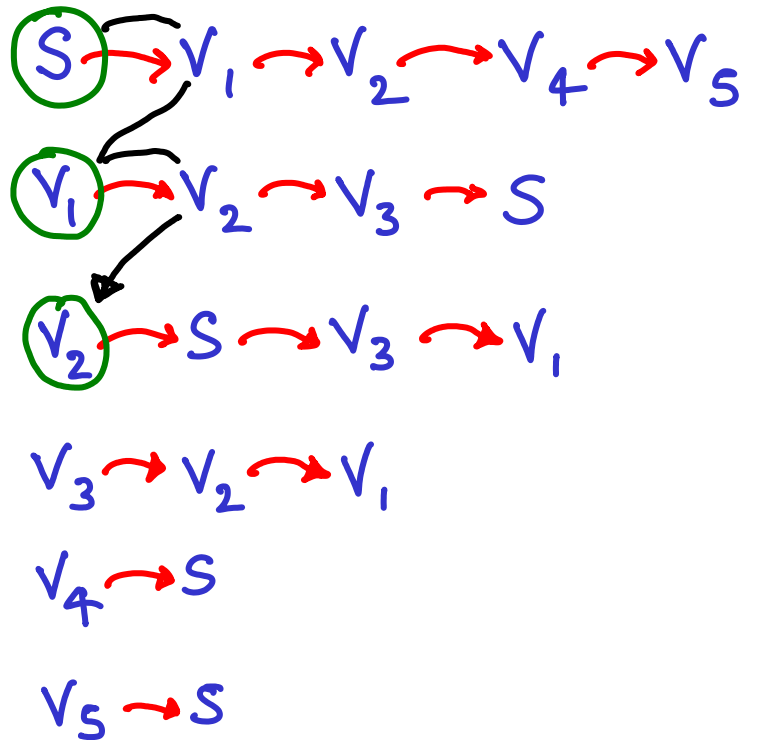
Adjacency list

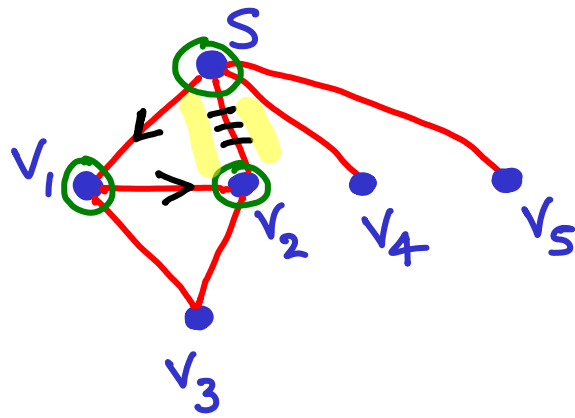




As with BFS, mark visited nodes.

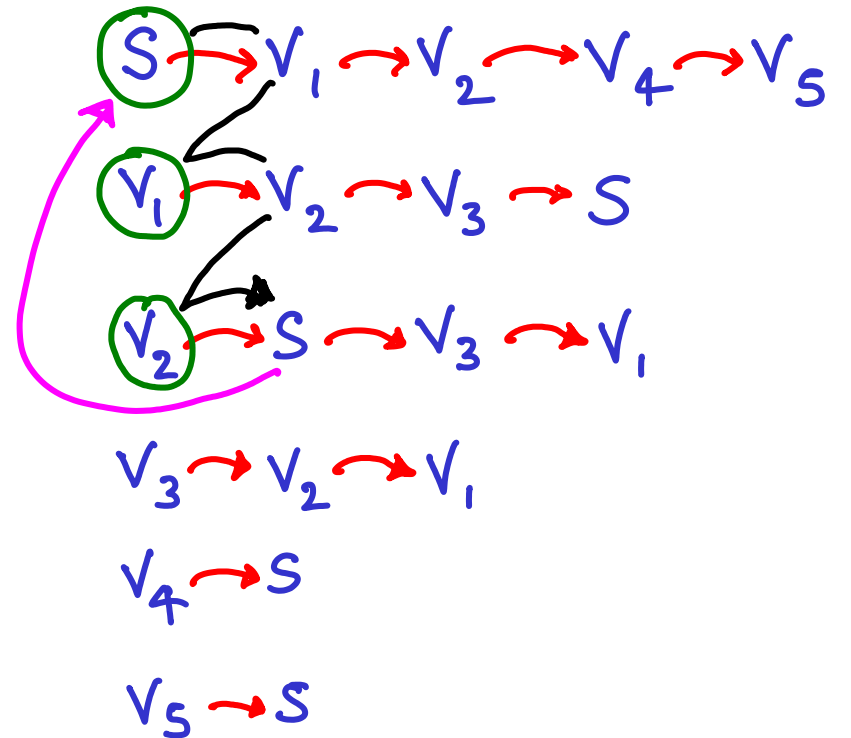
Adjacency list

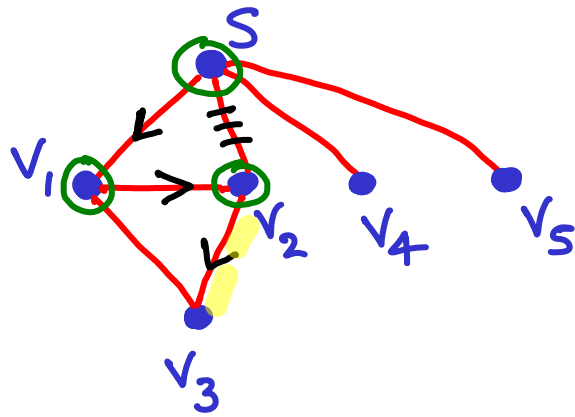




As with BFS, mark visited nodes.

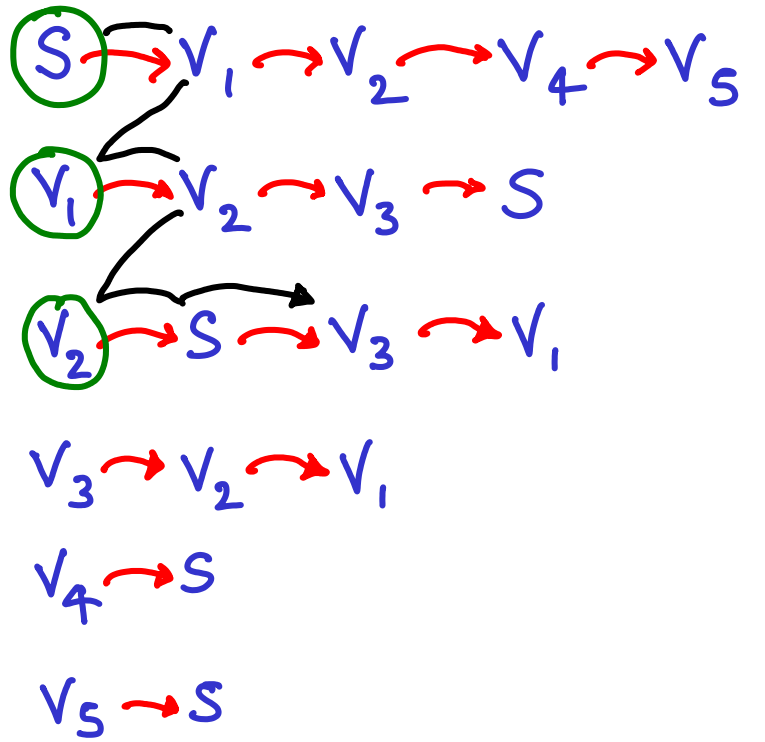
Adjacency list

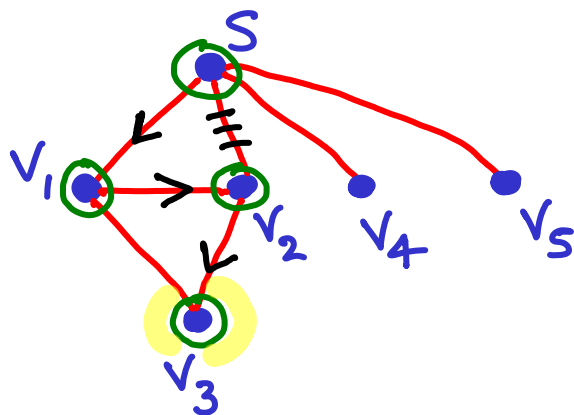




As with BFS, mark visited nodes.

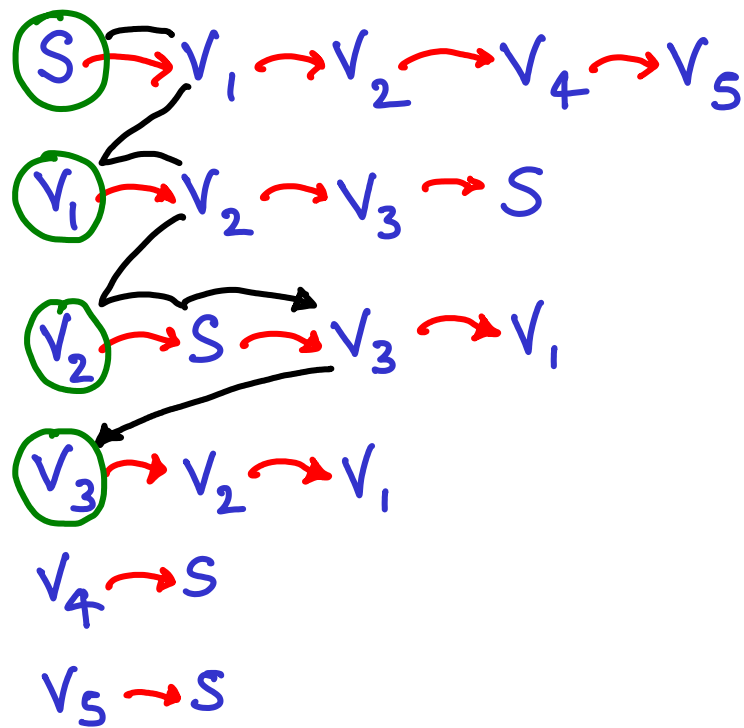
Adjacency list



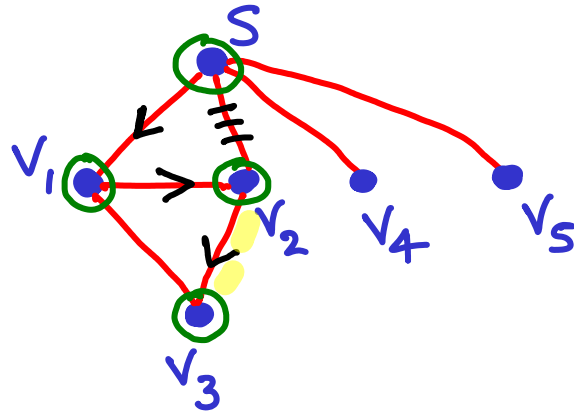


As with BFS, mark visited nodes.

Adjacency list

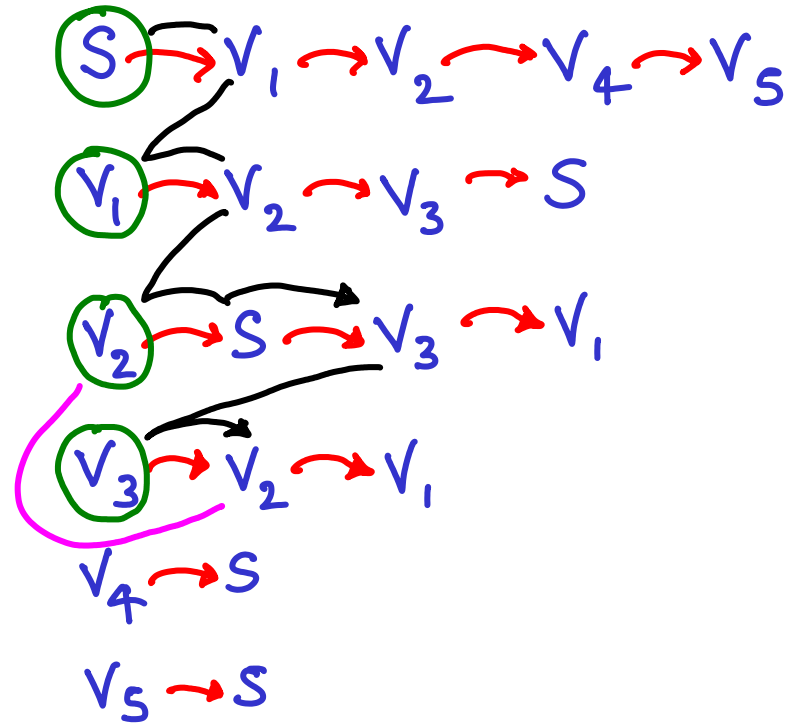


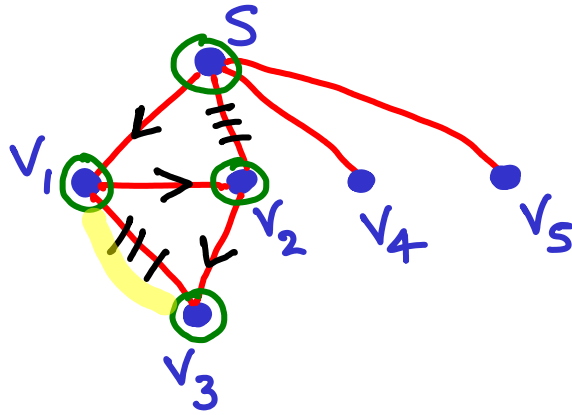




As with BFS, mark visited nodes.

Adjacency list

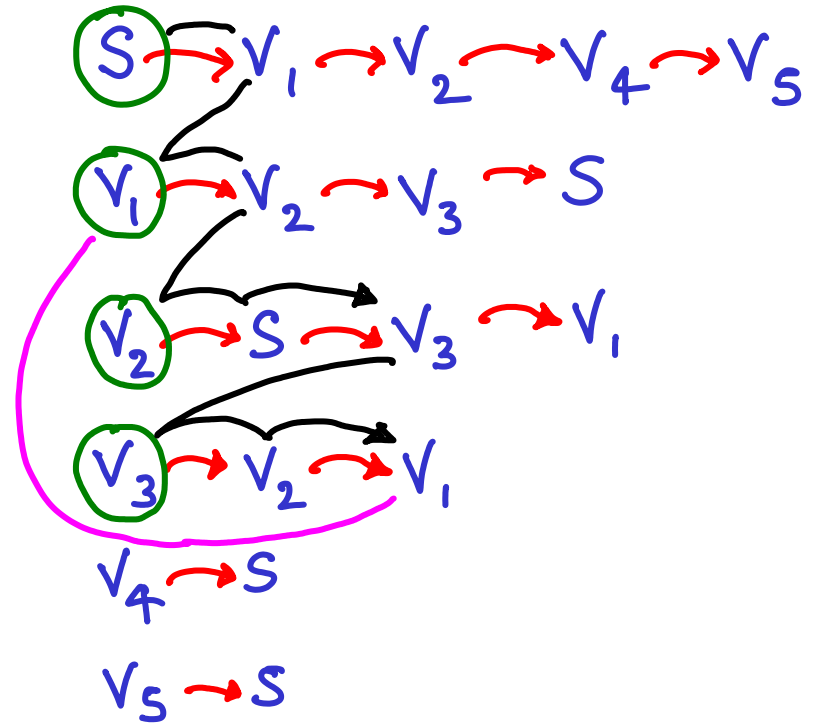


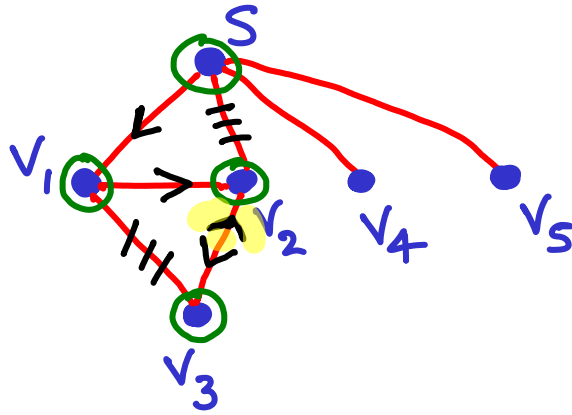


As with BFS, mark visited nodes.

$V_3$  has nowhere to go

Adjacency list

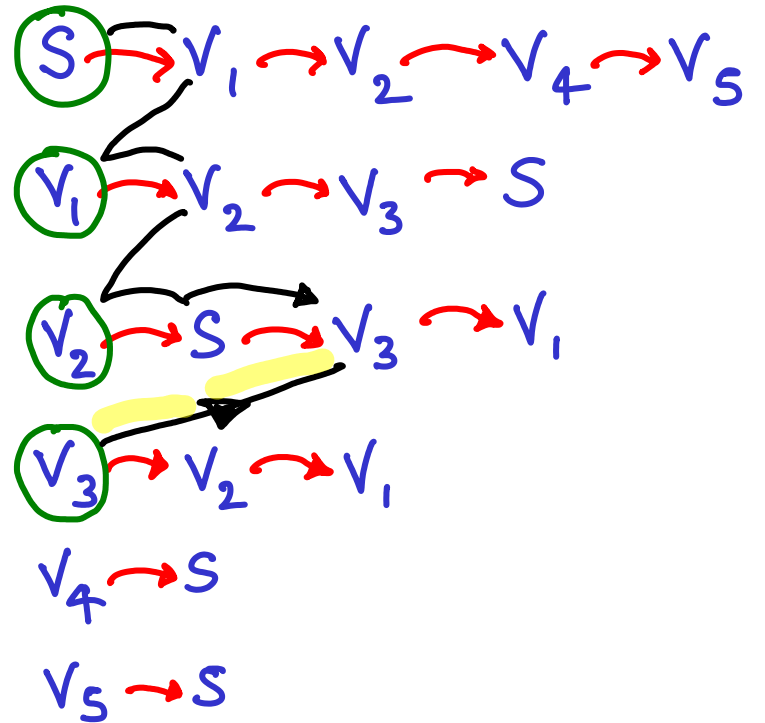


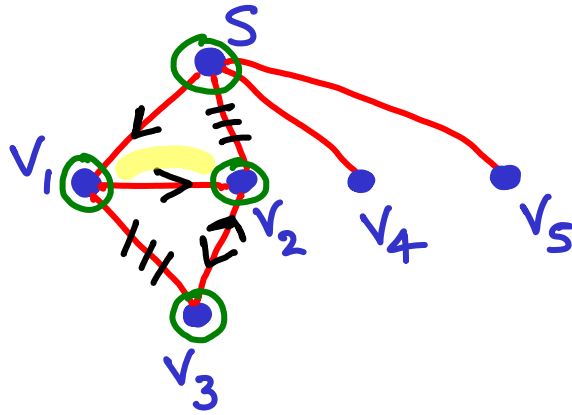


As with BFS, mark visited nodes.

$v_3$  came from  $Adj[v_2]$

Adjacency list

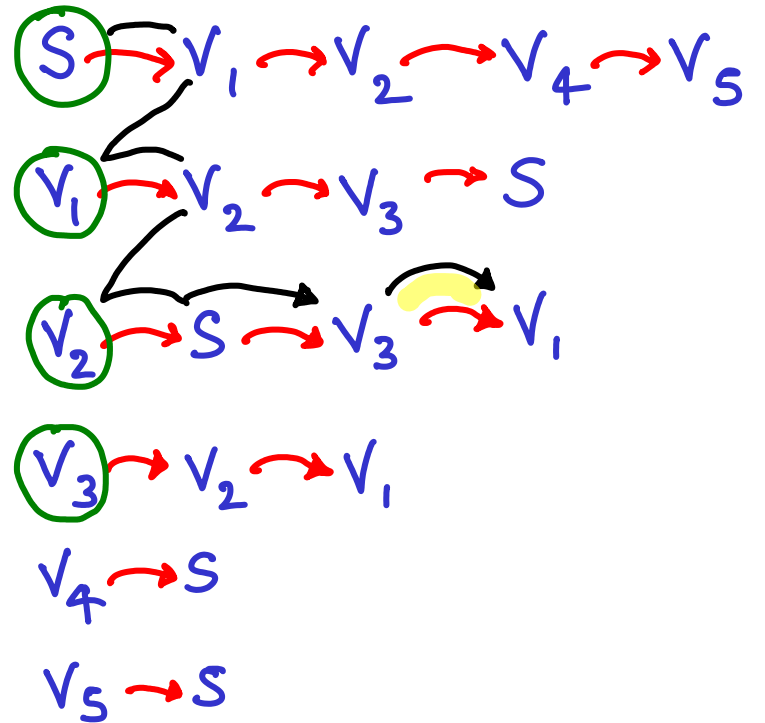


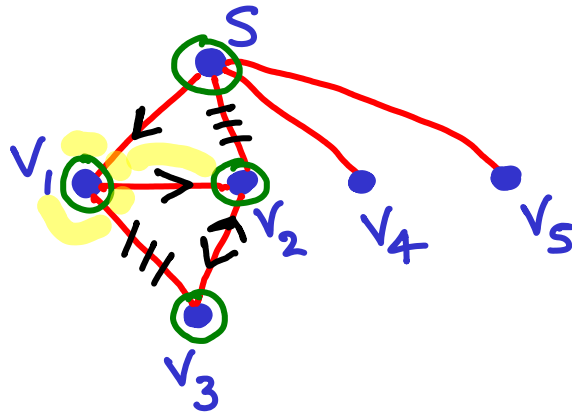


As with BFS, mark visited nodes.

$V_2$  continues its search

Adjacency list

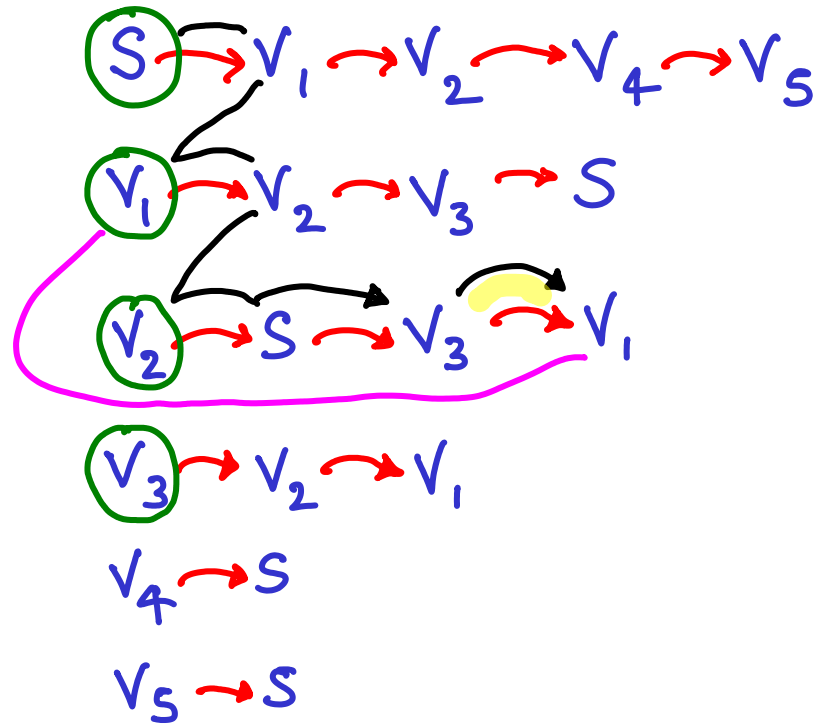


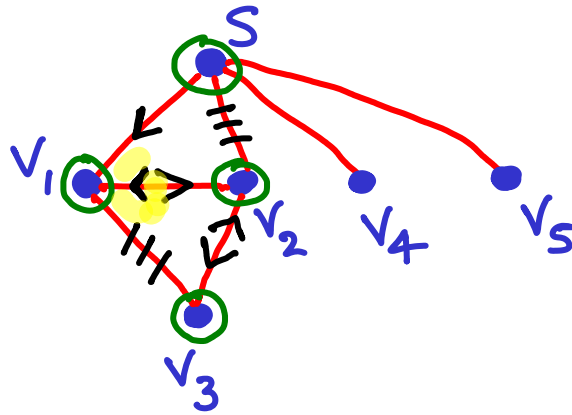


As with BFS, mark visited nodes.

$v_2$  continues its search  
...but  $v_1$  has been visited

Adjacency list



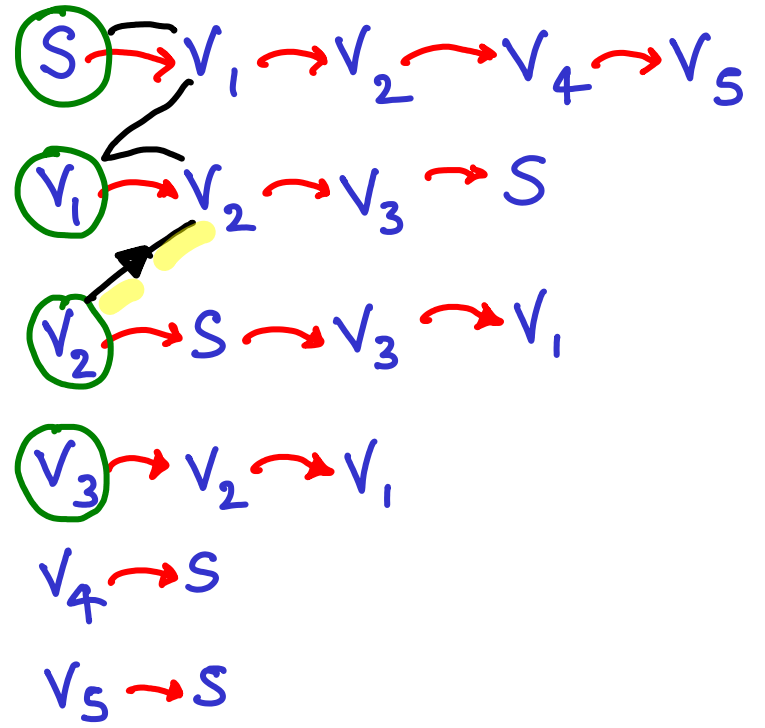


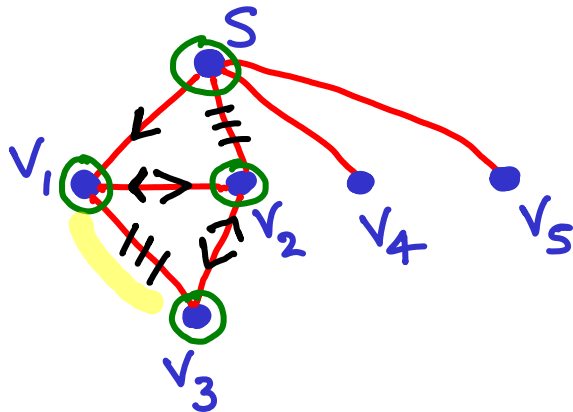
As with BFS, mark visited nodes.

Now  $v_2$  has nowhere to go.

$v_2$  came from  $Adj[v_1]$

Adjacency list

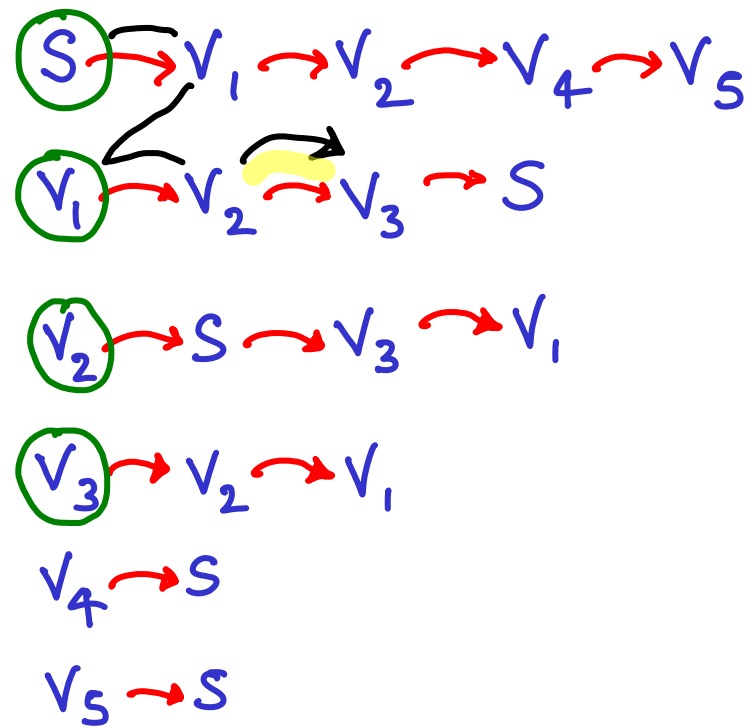


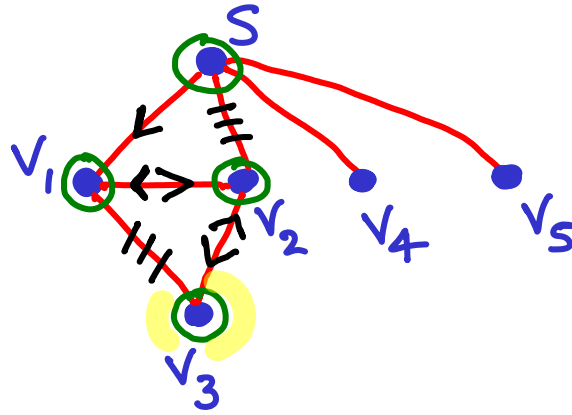


As with BFS, mark visited nodes.

$v_1$  doesn't know  $v_3$  is marked

Adjacency list

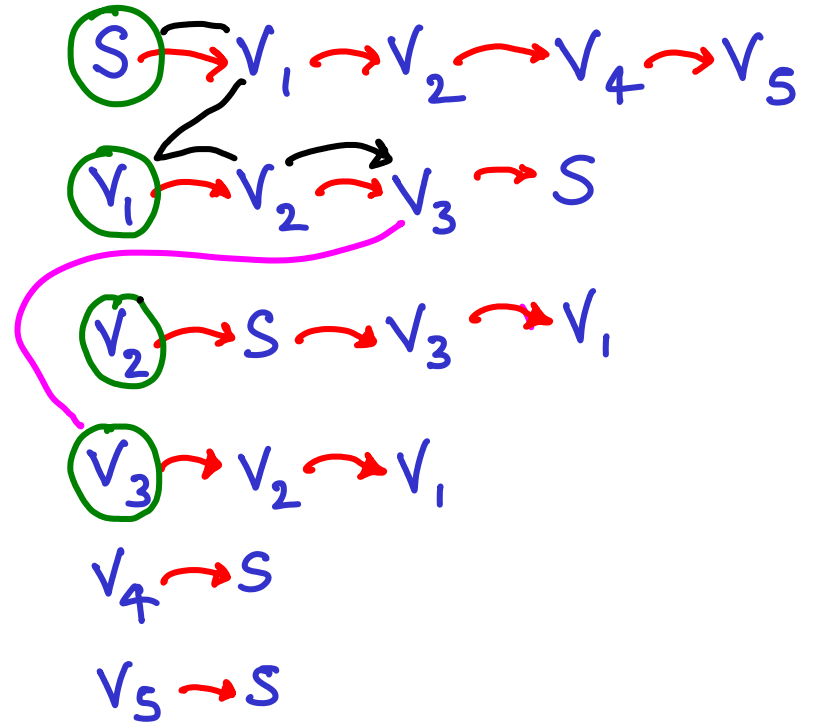




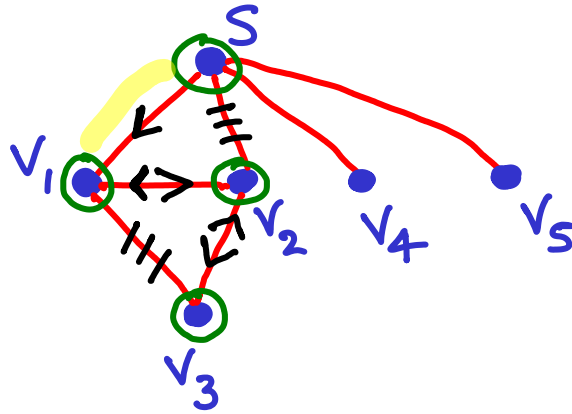
As with BFS, mark visited nodes.

$v_1$  discovers  $v_3$  is marked

Adjacency list



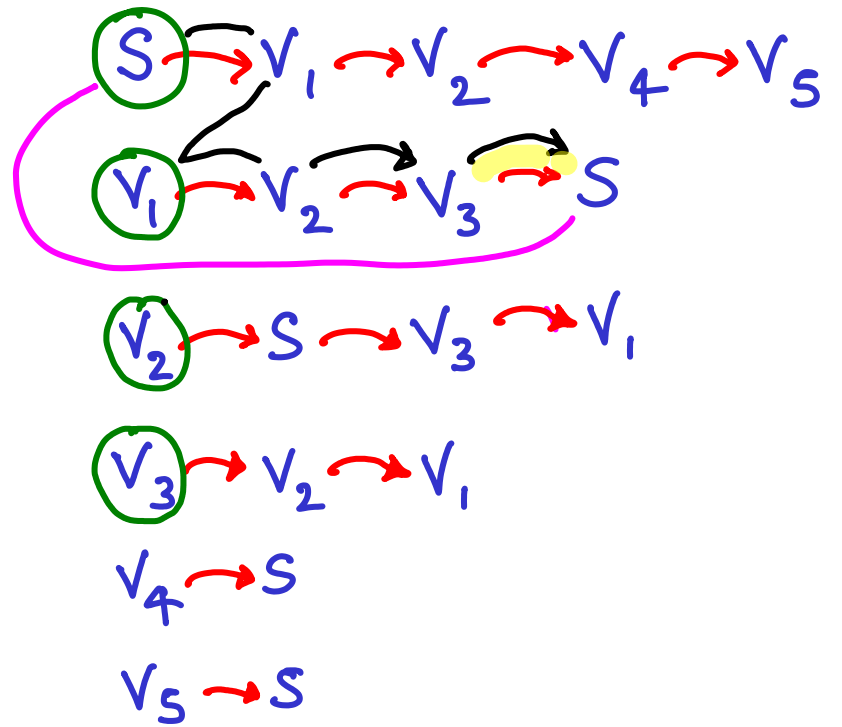


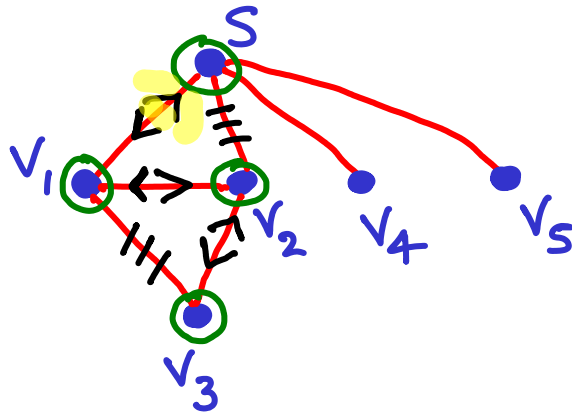


Adjacency list

As with BFS, mark visited nodes.

$v_1$  discovers  $S$  is marked and has nowhere else to go

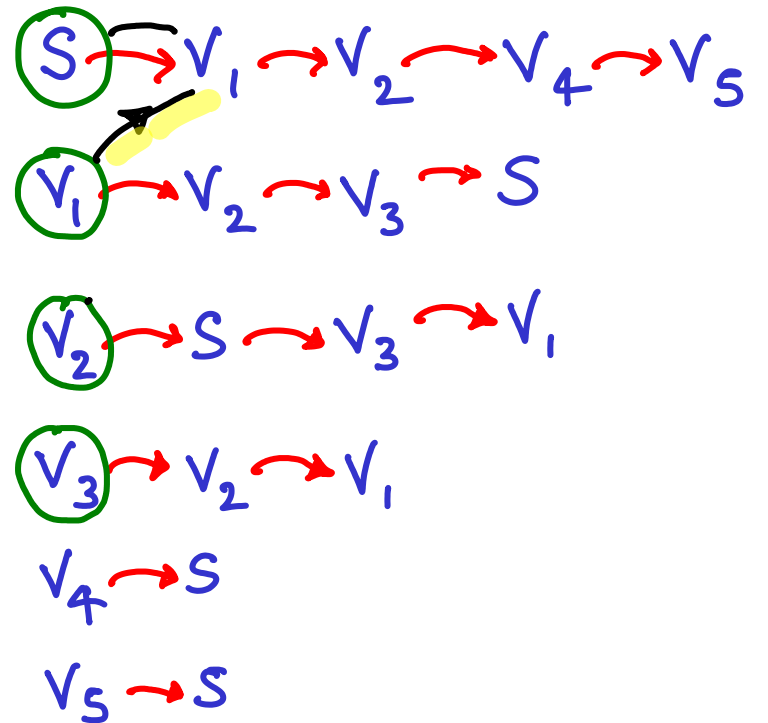


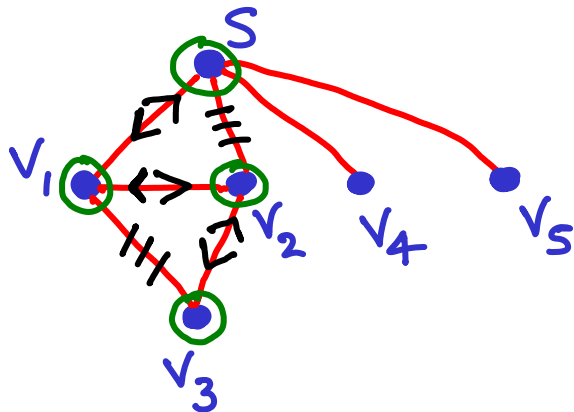


As with BFS, mark visited nodes.

$V_1$  came from  $Adj[S]$

Adjacency list





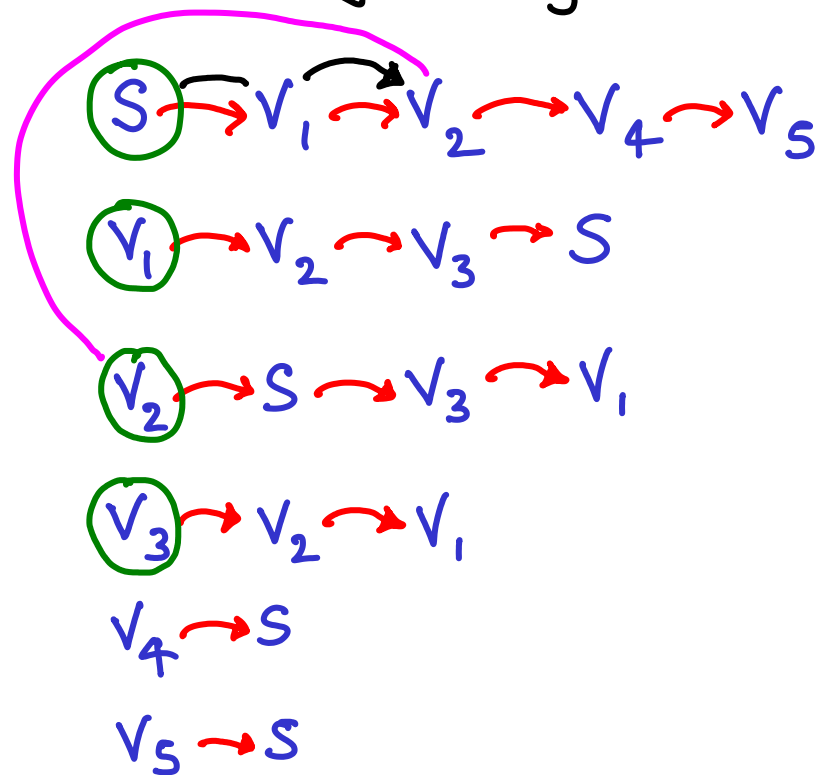
As with BFS, mark visited nodes.

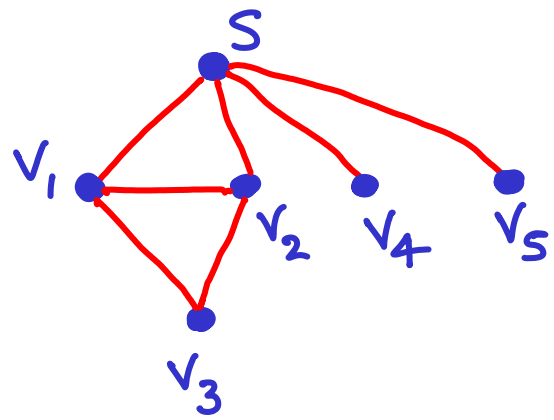
s continues on Adj[s]...

... discovers v<sub>2</sub> is marked

etc

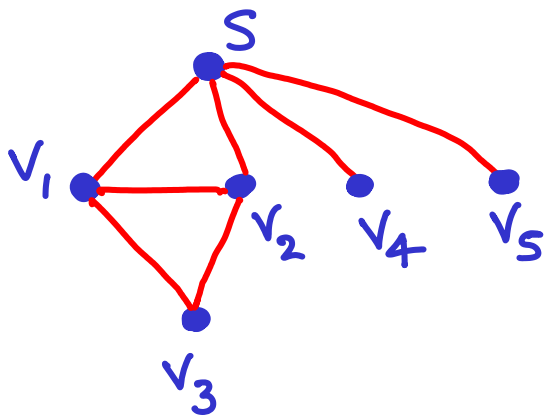
Adjacency list





DFS( $s$ )

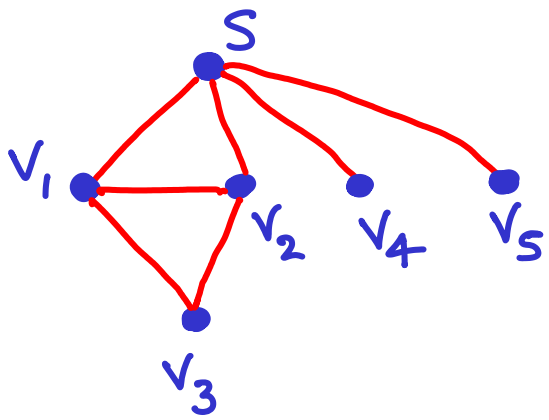
- mark  $s$



DFS(s)

- mark s

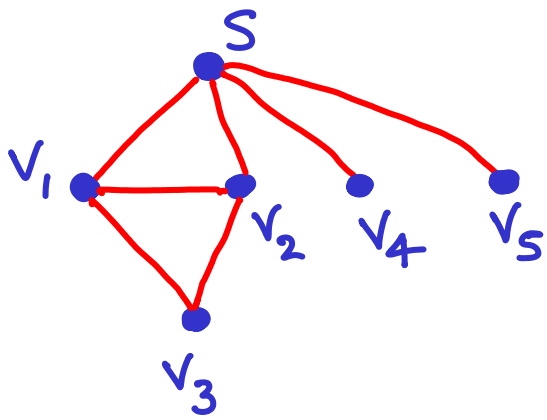
- for every neighbor  $v_i$  of s // i.e. scan  $Adj[s]$   
if  $v_i = t$ , DONE // only if looking for t



DFS(s)

- mark s

- for every neighbor  $v_i$  of s // i.e. scan  $Adj[s]$   
if  $v_i = t$ , DONE // only if looking for t  
if  $v_i$  is unmarked  
then ?

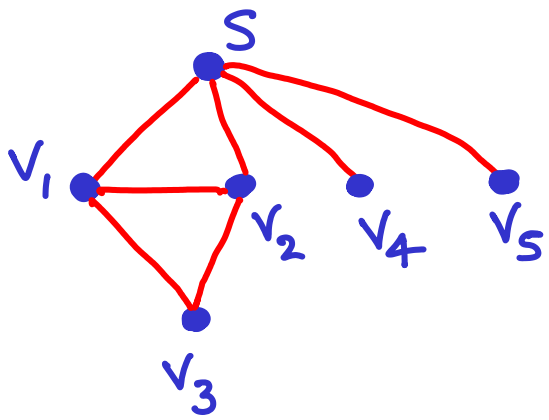


DFS(s)

- mark s

- for every neighbor  $v_i$  of s // i.e. scan  $Adj[s]$   
if  $v_i = t$ , DONE // only if looking for t  
if  $v_i$  is unmarked

- DFS( $v_i$ )



DFS(s)

- mark s

- for every neighbor  $v_i$  of s // i.e. scan  $Adj[s]$   
if  $v_i = t$ , DONE // only if looking for t  
if  $v_i$  is unmarked

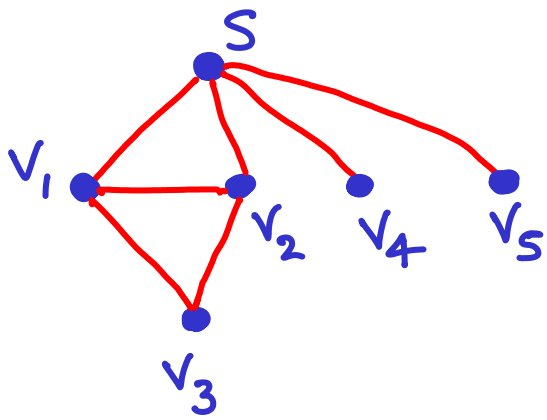
- set  $parent(v_i) \rightarrow s$

- set  $depth(v_i) \rightarrow 1 + depth(s)$

- DFS( $v_i$ )

} only if you  
want to keep  
the structure  
(init depth = 0)





DFS(s)

- mark s

- for every neighbor  $v_i$  of s // i.e. scan  $Adj[s]$   
if  $v_i = t$ , DONE // only if looking for t  
if  $v_i$  is unmarked

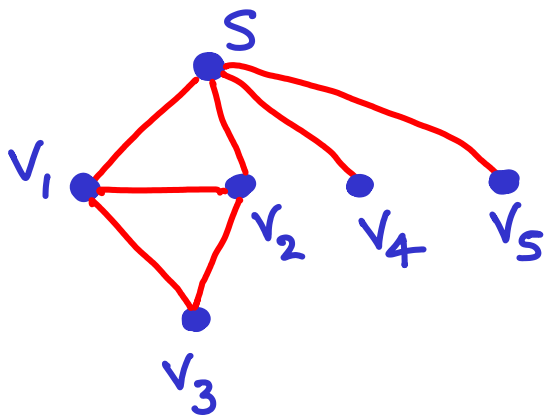
- set  $parent(v_i) \rightarrow s$

- set  $depth(v_i) \rightarrow 1 + depth(s)$

- DFS( $v_i$ )

} only if you  
want to keep  
the structure  
(init depth = 0)

time: ?



DFS(s)

- mark s

- for every neighbor  $v_i$  of s // i.e. scan  $Adj[s]$   
 if  $v_i = t$ , DONE // only if looking for t


if  $v_i$  is unmarked

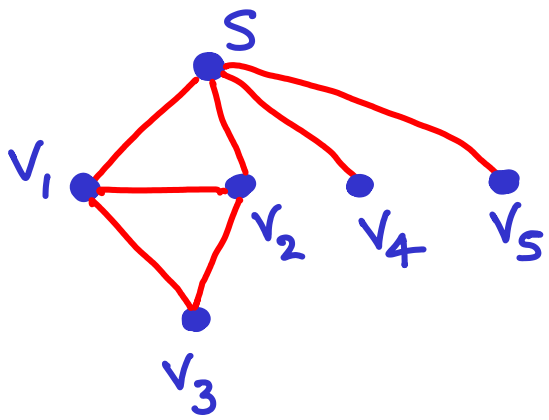
- set  $parent(v_i) \rightarrow s$

- set  $depth(v_i) \rightarrow 1 + depth(s)$

- DFS( $v_i$ )

} only if you  
 want to keep  
 the structure  
 (init depth = 0)

time:  $O(|E|)$  //  $|E|$ : size of component  
 (we use every edge twice  &  $E > V - 1$ )



DFS(s)

- mark s

- for every neighbor  $v_i$  of s // i.e. scan  $Adj[s]$   
 if  $v_i = t$ , DONE // only if looking for t

if  $v_i$  is unmarked


- set  $parent(v_i) \rightarrow s$

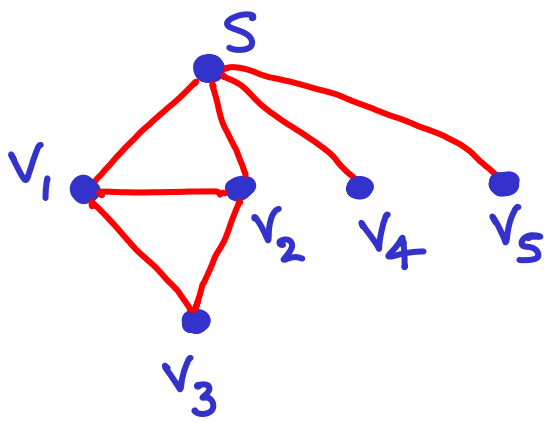
- set  $depth(v_i) \rightarrow 1 + depth(s)$

- DFS( $v_i$ )

} only if you want to keep the structure (init depth = 0)

Data structure?

time:  $O(|E|)$  //  $|E|$ : size of component  
 (we use every edge twice  &  $E > V - 1$ )



DFS(s)

- mark s

- for every neighbor  $v_i$  of s // i.e. scan  $Adj[s]$   
 if  $v_i = t$ , DONE // only if looking for t

if  $v_i$  is unmarked

- set  $parent(v_i) \rightarrow s$


- set  $depth(v_i) \rightarrow 1 + depth(s)$

- DFS( $v_i$ )

} only if you want to keep the structure (init depth = 0)

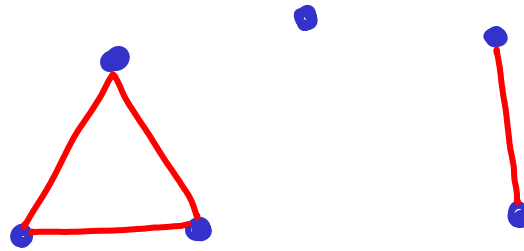
Data structure?

Stack

time:  $O(|E|)$  //  $|E|$ : size of component  
 (we use every edge twice  &  $E > V - 1$ )

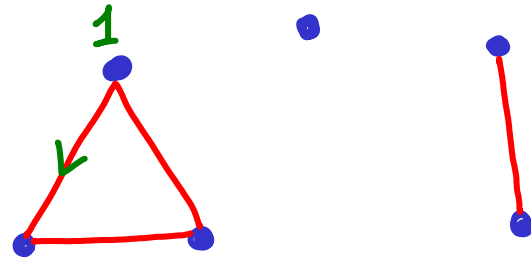
Dfs on a non-connected graph  $G$

For every vertex  $v_i$  in  $G$   
if  $v_i$  is unmarked  
DFS( $v_i$ )



# DFS on a non-connected graph $G$

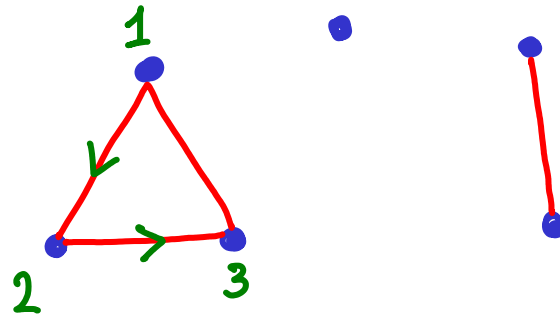
For every vertex  $v_i$  in  $G$   
if  $v_i$  is unmarked  
DFS( $v_i$ )



It is also easy to keep a counter to keep track  
of the "time" at which each vertex is  
first encountered & fully processed

# DFS on a non-connected graph $G$

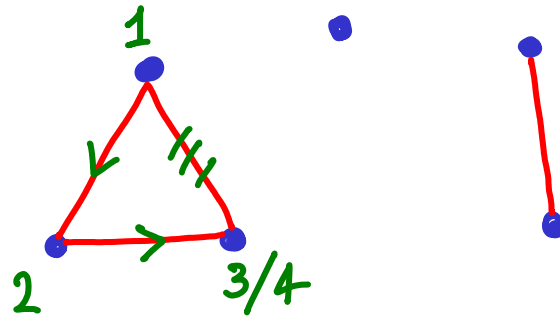
For every vertex  $v_i$  in  $G$   
if  $v_i$  is unmarked  
DFS( $v_i$ )



It is also easy to keep a counter to keep track  
of the "time" at which each vertex is  
first encountered & fully processed

# DFS on a non-connected graph $G$

For every vertex  $v_i$  in  $G$   
if  $v_i$  is unmarked  
DFS( $v_i$ )

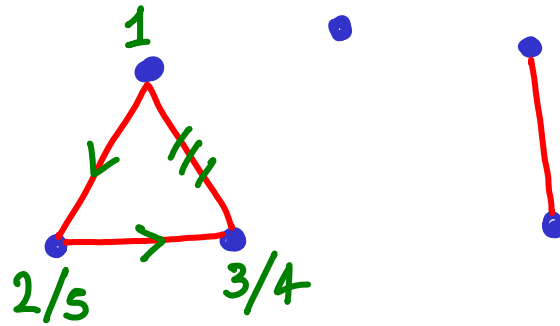


It is also easy to keep a counter to keep track  
of the "time" at which each vertex is  
first encountered & fully processed



# DFS on a non-connected graph $G$

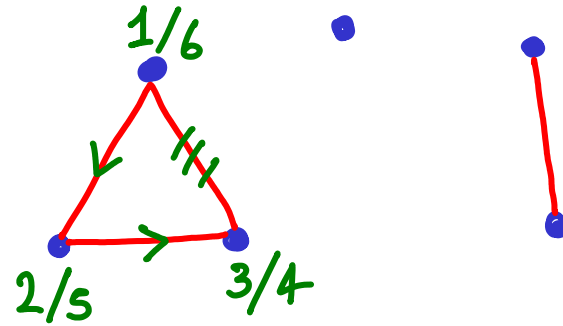
For every vertex  $v_i$  in  $G$   
if  $v_i$  is unmarked  
DFS( $v_i$ )



It is also easy to keep a counter to keep track  
of the "time" at which each vertex is  
first encountered & fully processed

# DFS on a non-connected graph $G$

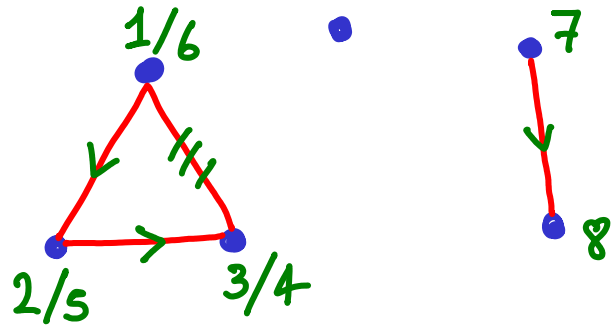
For every vertex  $v_i$  in  $G$   
if  $v_i$  is unmarked  
DFS( $v_i$ )



It is also easy to keep a counter to keep track  
of the "time" at which each vertex is  
first encountered & fully processed

# DFS on a non-connected graph $G$

For every vertex  $v_i$  in  $G$   
if  $v_i$  is unmarked  
DFS( $v_i$ )

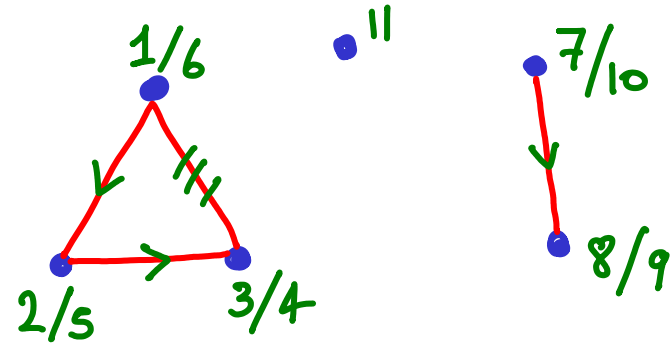


It is also easy to keep a counter to keep track of the "time" at which each vertex is first encountered & fully processed

DFS on a non-connected graph  $G$

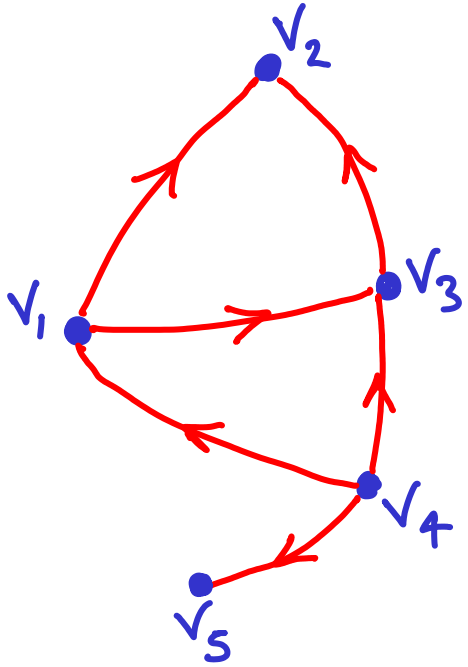
time:  $\mathcal{O}(V+E)$

For every vertex  $v_i$  in  $G$   
if  $v_i$  is unmarked  
DFS( $v_i$ )

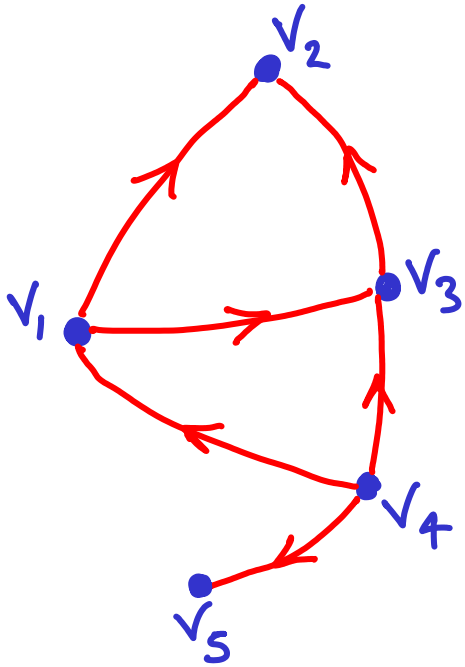


It is also easy to keep a counter to keep track  
of the "time" at which each vertex is  
first encountered & fully processed

DFS on a directed graph : similar to non-connected  
(process all vertices)



DFS on a directed graph : similar to non-connected  
(process all vertices)



$v_1$

$v_2$

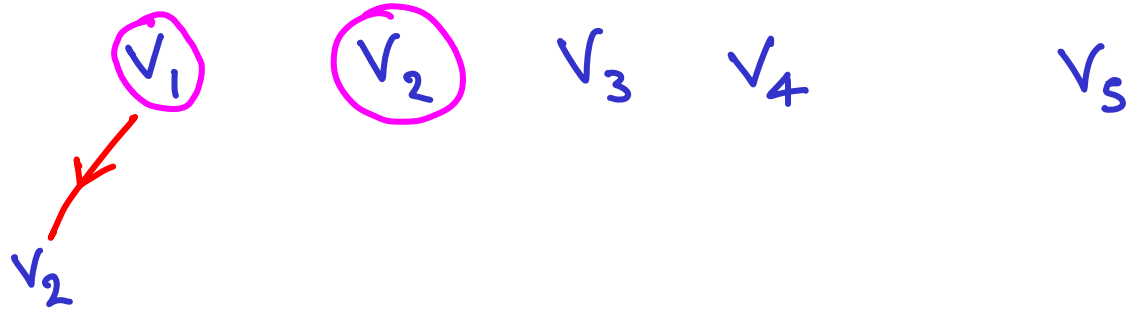
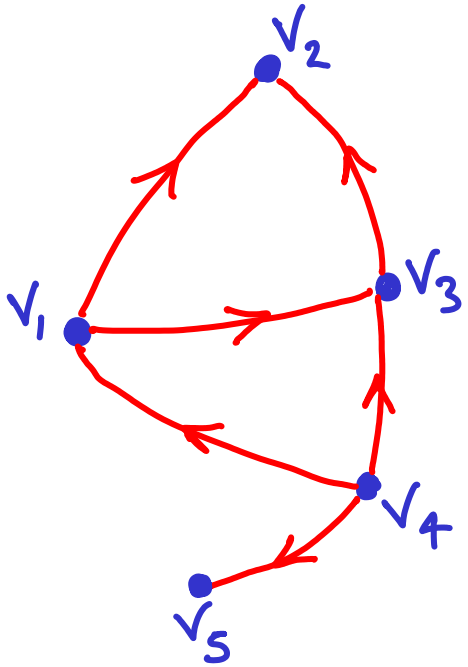
$v_3$

$v_4$

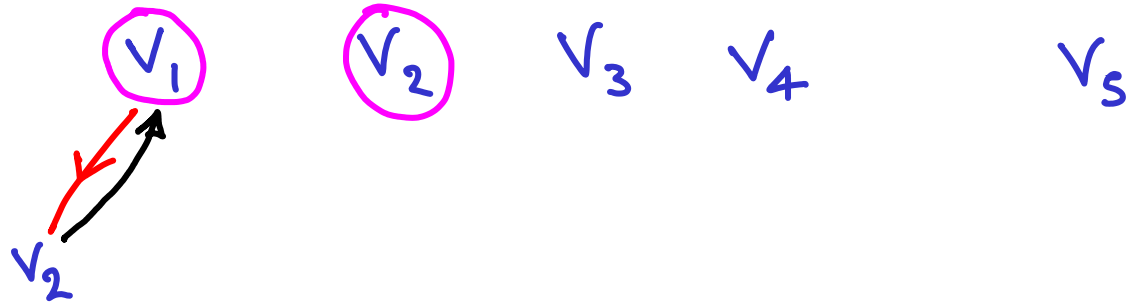
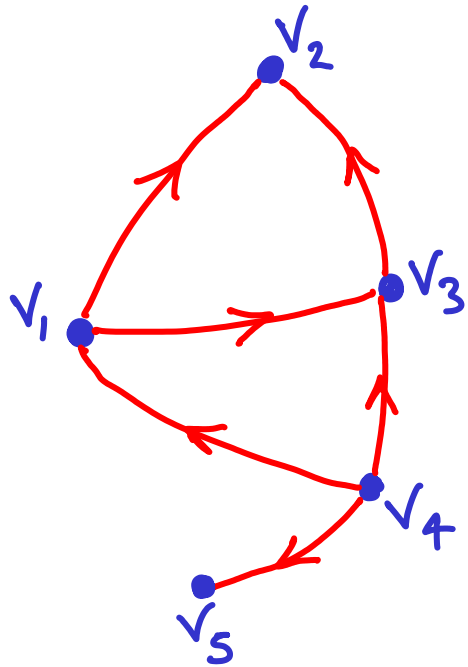
$v_5$

iteratively, mark & explore if unmarked

DFS on a directed graph : similar to non-connected  
(process all vertices)

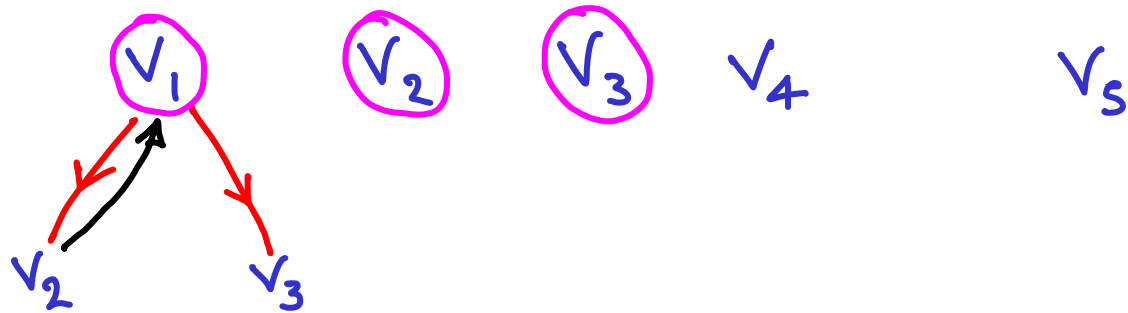
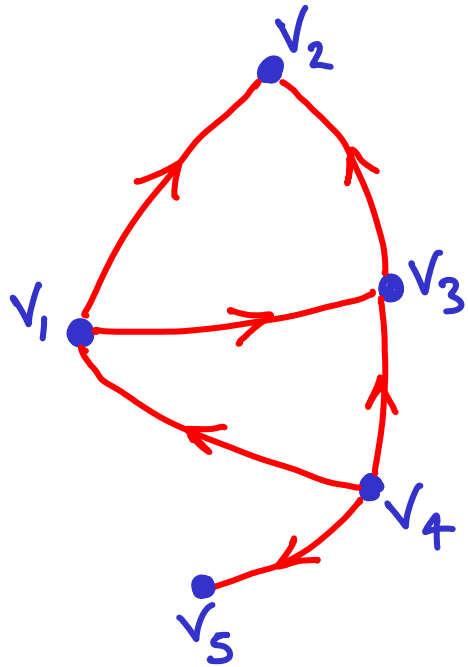


DFS on a directed graph : similar to non-connected  
(process all vertices)

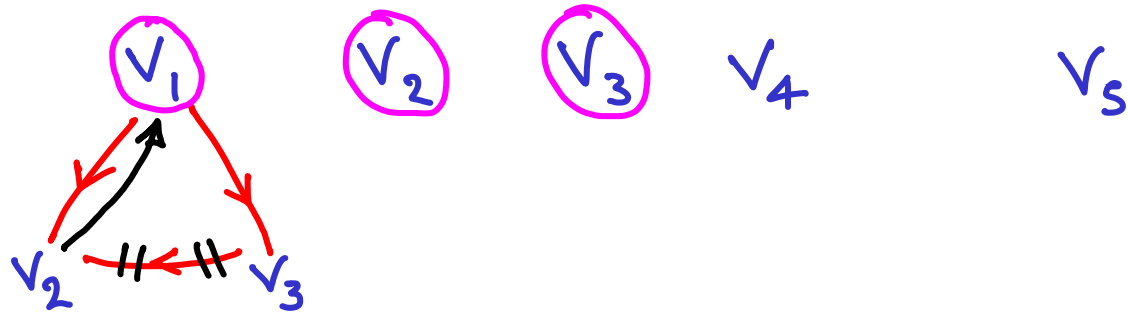
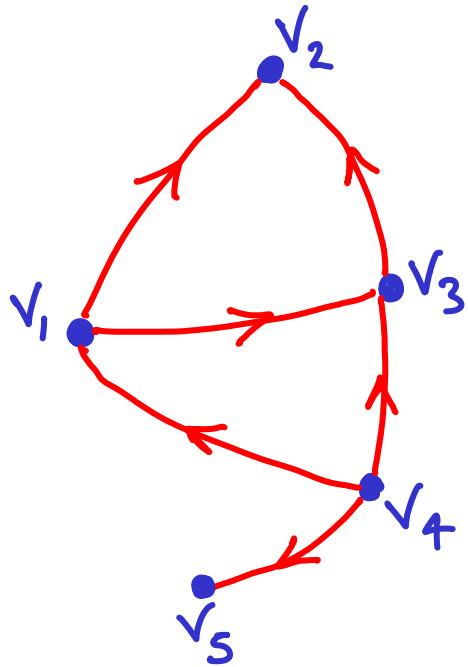




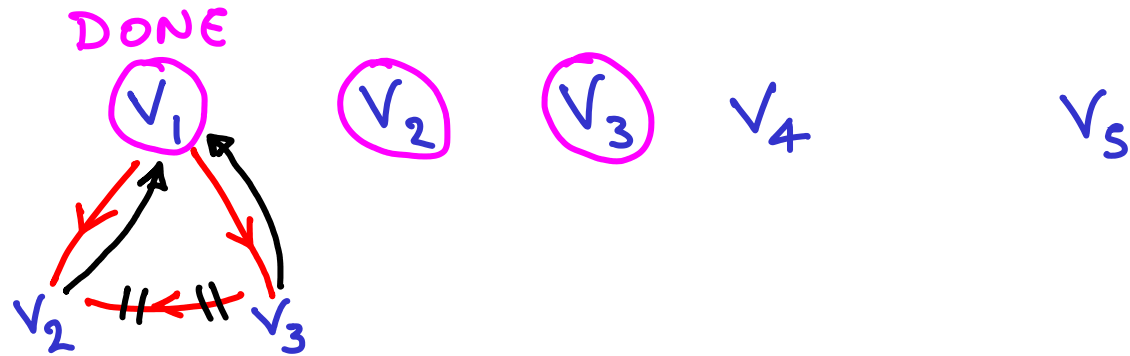
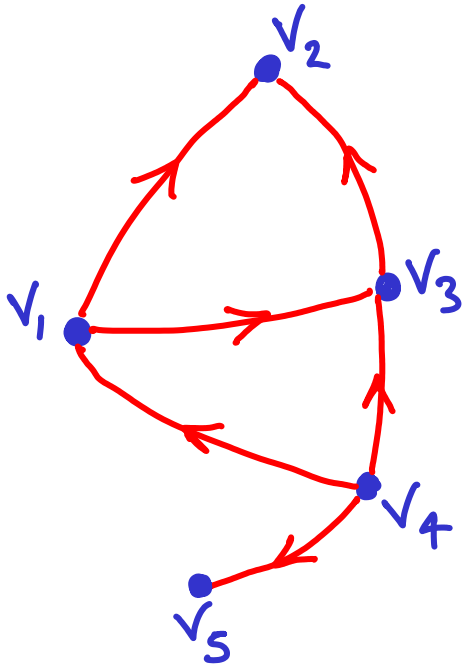
DFS on a directed graph : similar to non-connected  
(process all vertices)



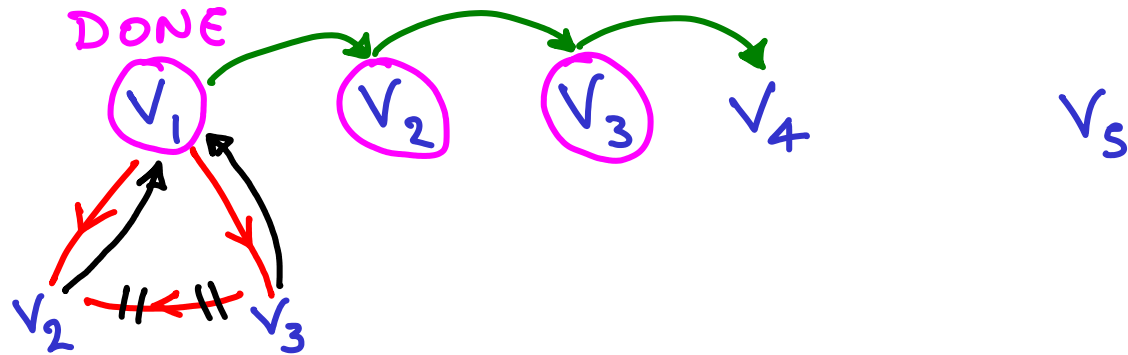
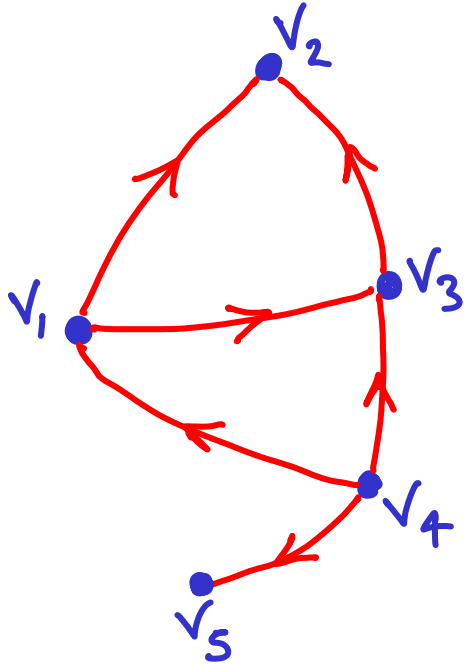
DFS on a directed graph : similar to non-connected  
(process all vertices)



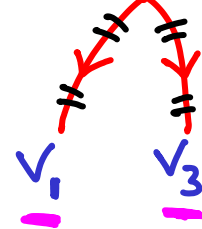
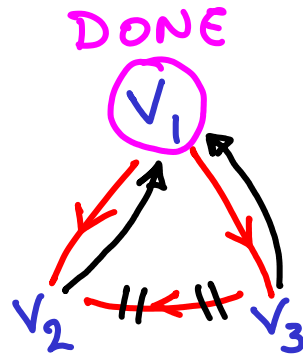
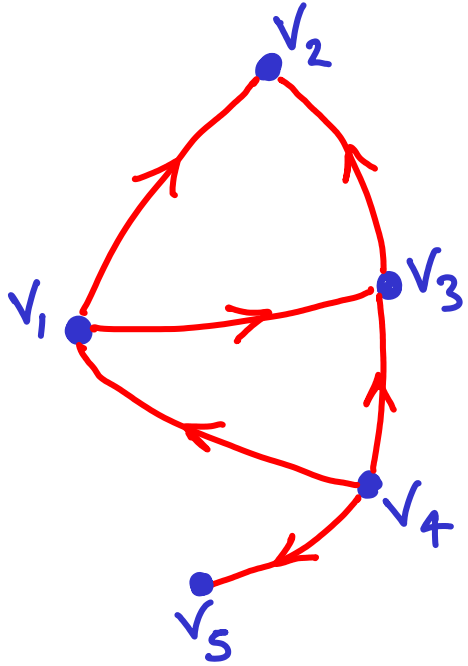
DFS on a directed graph : similar to non-connected  
(process all vertices)



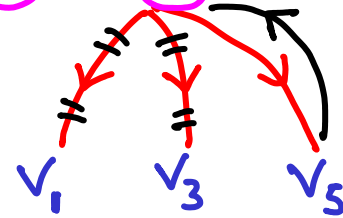
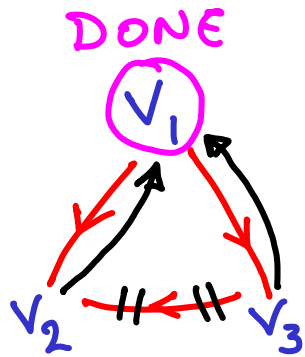
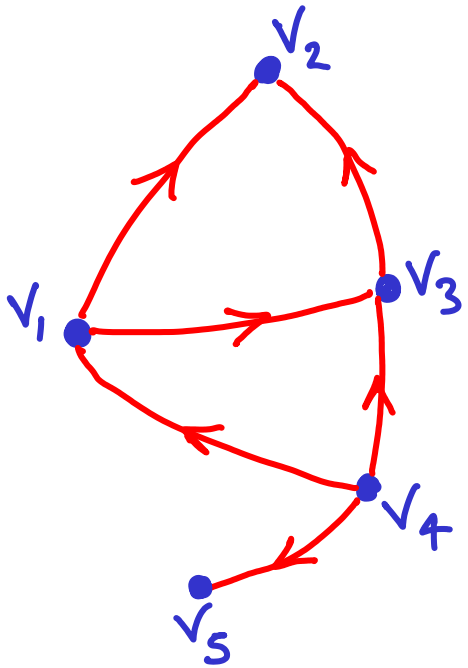
DFS on a directed graph : similar to non-connected  
(process all vertices)



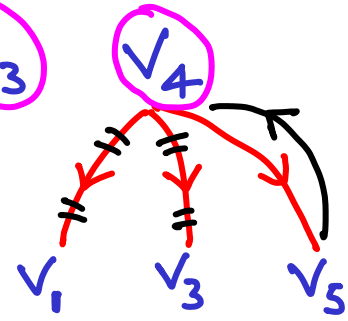
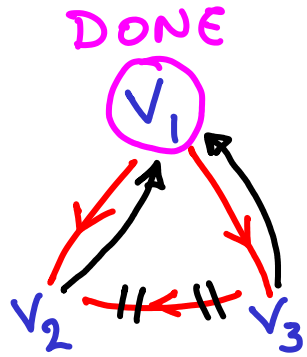
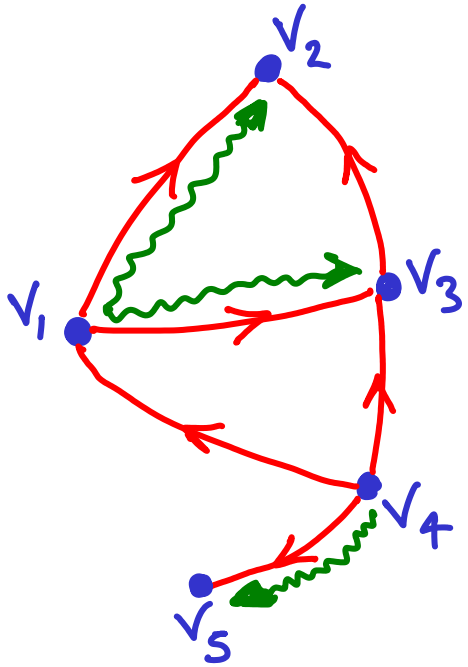
DFS on a directed graph : similar to non-connected  
(process all vertices)



DFS on a directed graph : similar to non-connected  
(process all vertices)

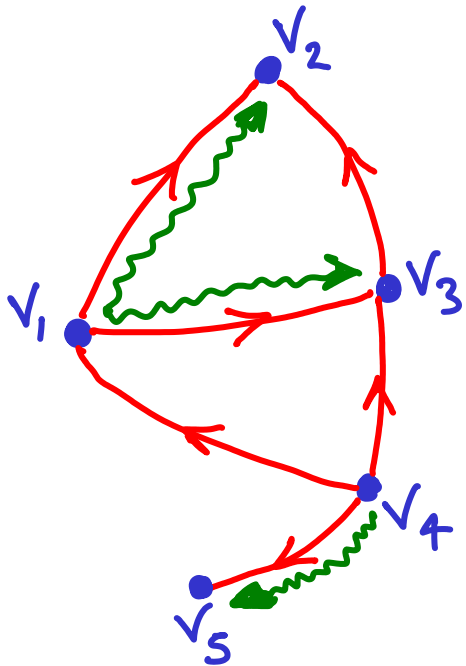


DFS on a directed graph : similar to non-connected  
(process all vertices)



DFS on a directed graph : similar to non-connected  
(process all vertices)

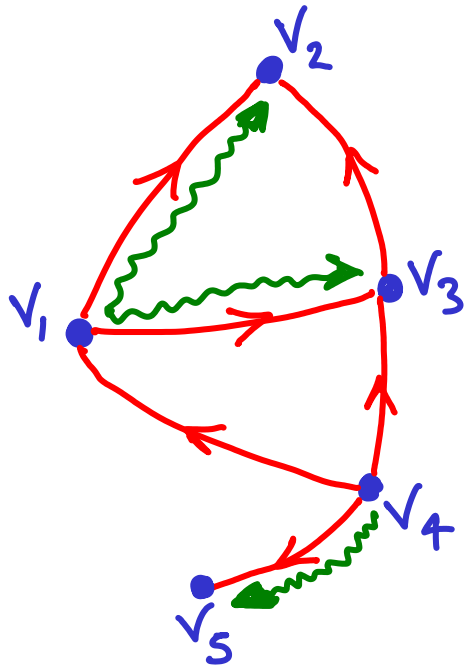
1, 2, 3, 4, 5



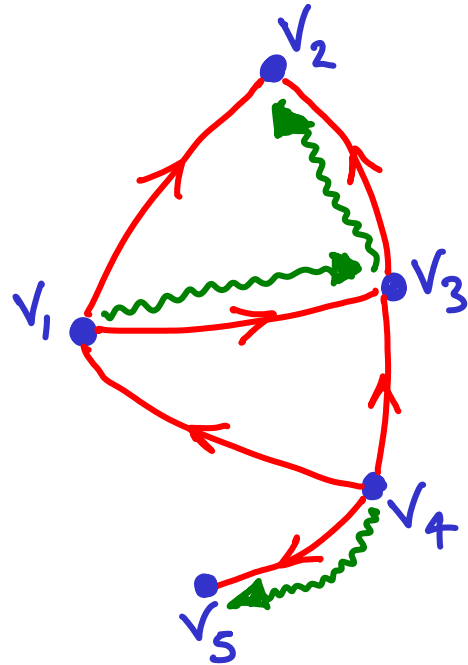


DFS on a directed graph : similar to non-connected  
(process all vertices)

1, 2, 3, 4, 5



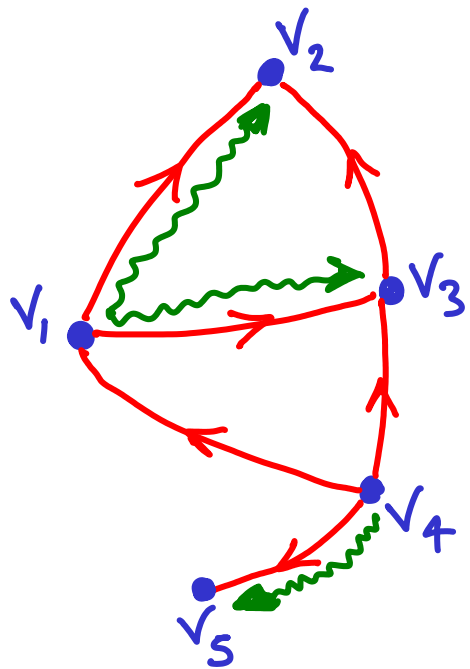
again  
1, 2, 3, 4, 5



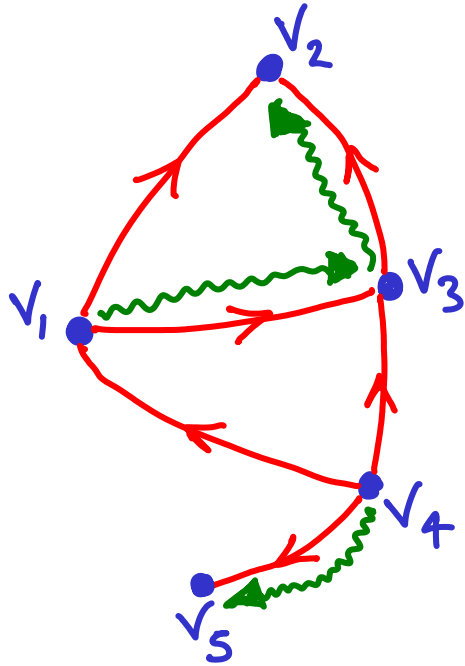
if  $v_3$  came before  $v_2$   
in  $Adj[v_1]$

DFS on a directed graph : similar to non-connected  
(process all vertices)

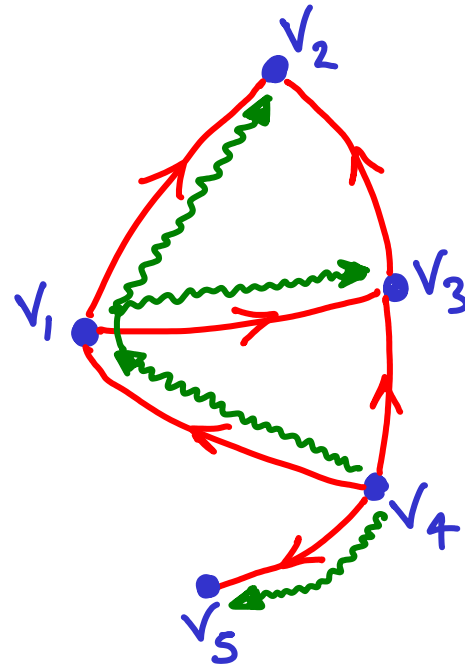
1, 2, 3, 4, 5



again  
1, 2, 3, 4, 5

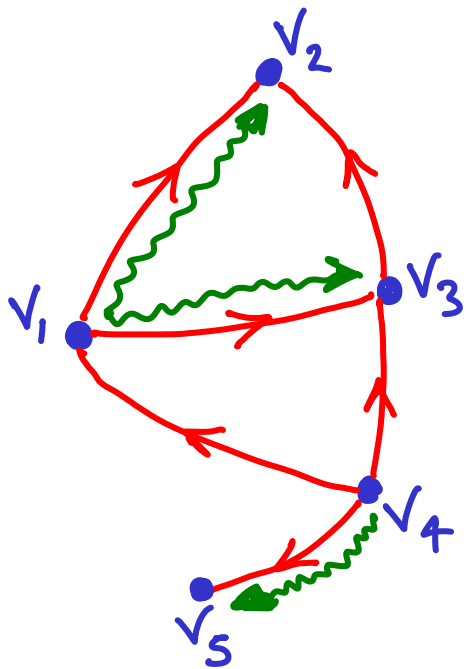


order of search?

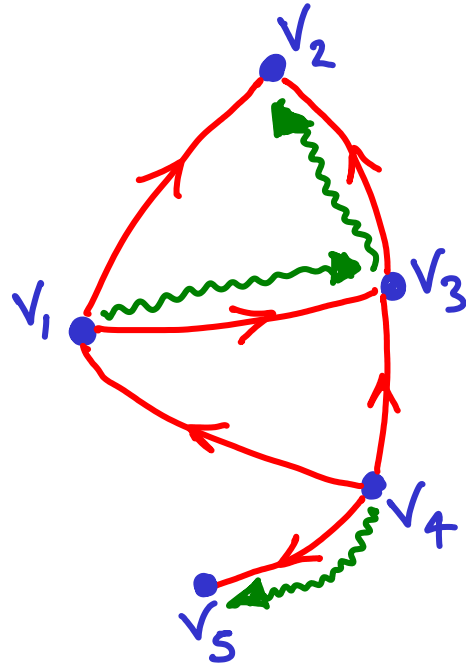


DFS on a directed graph : similar to non-connected  
(process all vertices)

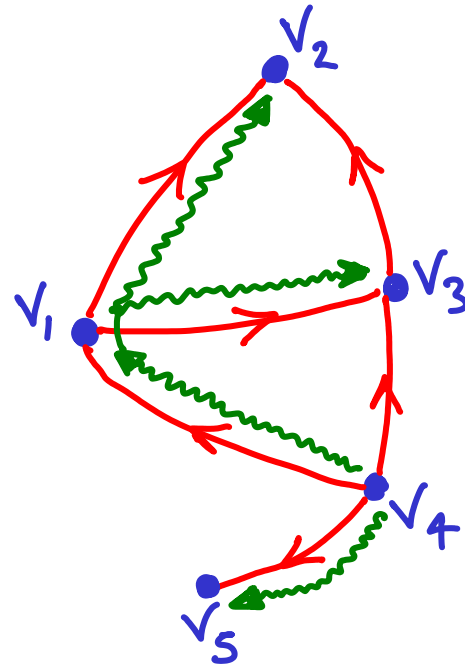
1, 2, 3, 4, 5



again  
1, 2, 3, 4, 5

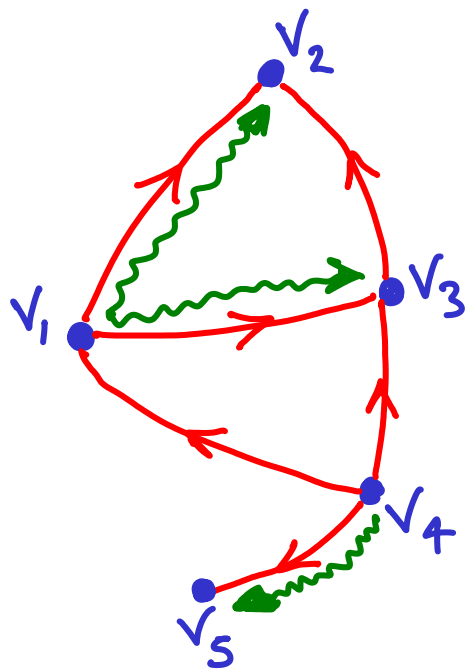


4 before 1, 5  
... before 3?

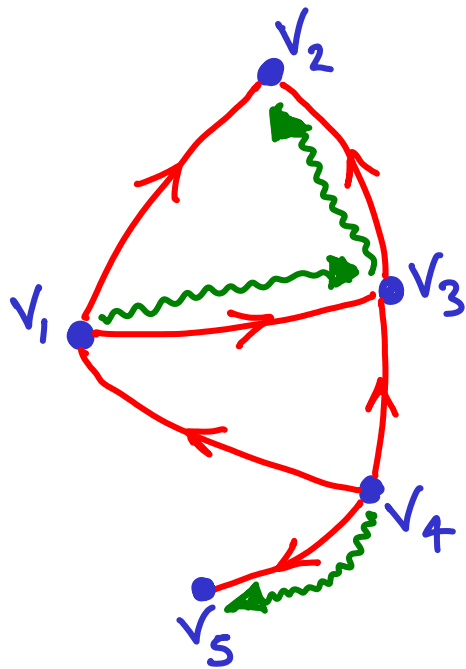


DFS on a directed graph : similar to non-connected  
(process all vertices)

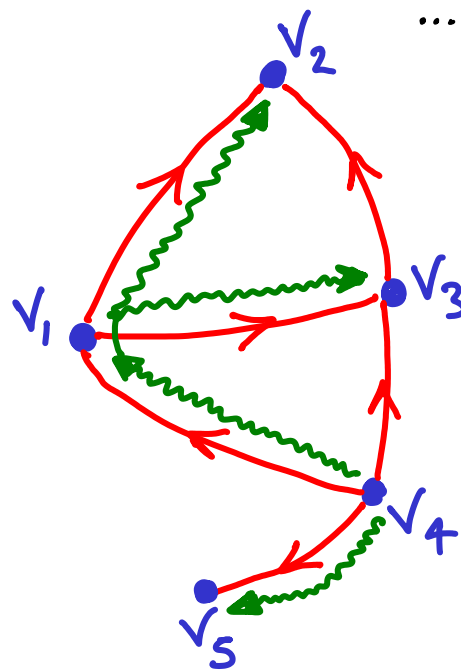
1, 2, 3, 4, 5



again  
1, 2, 3, 4, 5



4 before 1, 5  
... before 3  
...etc



3 before 1 in  
Adj[4]

