DEPTH FIRST SEARCH (DFS) - The greedy way to search
DEPTH FIRST SEARCH (DFS)

Follow an unvisited path for as long as possible.
DEPTH FIRST SEARCH (DFS)

Follow an unvisited path for as long as possible.
DEPTH FIRST SEARCH (DFS)

Follow an unvisited path for as long as possible.
DEPTH FIRST SEARCH (DFS)

Follow an unvisited path for as long as possible.
When you reach a vertex with only previously-visited neighbors,
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 with 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.
just redrawing
Adjacency list

S → V_1 → V_2 → V_4 → V_5
V_1 → V_2 → V_3 → S
V_2 → S → V_3 → V_1
V_3 → V_2 → V_1
V_4 → S
V_5 → S
As with BFS, mark visited nodes.

Adjacency list

\[
\begin{align*}
S & \rightarrow V_1 \\
V_1 & \rightarrow V_2 \\
V_2 & \rightarrow V_3 \\
V_3 & \rightarrow V_2 \\
V_4 & \rightarrow S \\
V_5 & \rightarrow S
\end{align*}
\]
As with BFS, mark visited nodes.
As with BFS, mark visited nodes.
As with BFS, mark visited nodes.
As with BFS, mark visited nodes.
As with BFS, mark visited nodes.
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

- \( S \rightarrow V_1 \rightarrow V_2 \rightarrow V_4 \rightarrow V_5 \)
- \( V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow S \)
- \( V_2 \rightarrow S \rightarrow V_3 \rightarrow V_1 \)
- \( V_3 \rightarrow V_2 \rightarrow V_1 \)
- \( V_4 \rightarrow S \)
- \( V_5 \rightarrow S \)
As with BFS, mark visited nodes.
As with BFS, mark visited nodes.

$V_3$ has nowhere to go.
As with BFS, mark visited nodes.

$V_3$ came from $\text{Adj}[v_2]$.  

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.

\[ V_2 \text{ continues its search} \]

Adjacency list:

\[
\begin{align*}
S & \rightarrow V_1 \rightarrow V_2 \rightarrow V_4 \rightarrow V_5 \\
V_1 & \rightarrow V_2 \rightarrow V_3 \rightarrow S \\
V_2 & \rightarrow S \rightarrow V_3 \rightarrow V_1 \\
V_3 & \rightarrow V_2 \rightarrow V_1 \\
V_4 & \rightarrow S \\
V_5 & \rightarrow S
\end{align*}
\]
As with BFS, mark visited nodes.

V₂ continues its search
...but V₁ has been visited
As with BFS, mark visited nodes.

Now $v_2$ has nowhere to go. $v_2$ came from $\text{Adj}[v_1]$. 
As with BFS, mark visited nodes.

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

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.

$v_1$ discovers $v_3$ is marked.

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.

\( V_1 \) discovers \( S \) is marked and has nowhere else to go.

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.

$V_1$ came from Adj[$S$]

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.

S continues on $\text{Adj}[S]$...

...discovers $v_2$ is marked

etc
explore one connected component

\text{DFS}(u) \quad /\quad \text{DFS starting at } u
DFS(u)  // DFS starting at u
for every neighbor \(v_i\) of u  // i.e. scan \(\text{Adj}[u]\)
DFS(u)  // DFS starting at u
for every neighbor v_i of u  // i.e. scan Adj[u]
if v_i is unmarked
DFS(u)  // DFS starting at u

for every neighbor v_i of u  // i.e. scan Adj[u]

if v_i is unmarked

mark v_i

set parent(v_i) \rightarrow u

set depth(v_i) \rightarrow 1 + \text{depth}(u)

} only if you want to keep the structure
DFS(u)    // DFS starting at u
for every neighbor v_i of u    // i.e. scan Adj[u]
    if v_i is unmarked
        mark v_i;
        set parent(v_i) → u
        set depth(v_i) → 1 + depth(u)
    DFS(v_i)
Find \( t \) from \( s \):

Start with:

- mark \( s \)
- \( \text{depth}(s) = 0 \)
- \( \text{DFS}(s) \)

\( \text{DFS}(u) \)  // DFS starting at \( u \)

- for every neighbor \( v_i \) of \( u \)  // i.e. scan \( \text{Adj}[u] \)
  - if \( v_i = t \), DONE
  - if \( v_i \) is unmarked
    - mark \( v_i \)
    - set \( \text{parent}(v_i) \rightarrow u \)
    - set \( \text{depth}(v_i) \rightarrow 1 + \text{depth}(u) \)
    - \( \text{DFS}(v_i) \)
Start with: mark s
depth(s) = 0
DFS(s)

time for connected G?

DFS(u) // DFS starting at u
for every neighbor vi of u // i.e. scan Adj[u]
  if vi = t, DONE
  if vi is unmarked
    mark vi
    set parent(vi) → u
    set depth(vi) → 1 + depth(u)
    DFS(vi)

} only if you want to keep the structure
Start with:
  mark s
  depth(s) = 0
  DFS(s)

\( O(1) \) 

if not connected?

DFS(u)  // DFS starting at u

for every neighbor \( v_i \) of u  // i.e. scan Adj[u]
  if \( v_i = t \), DONE
  if \( v_i \) is unmarked
    mark \( v_i \);
    set parent(\( v_i \)) \rightarrow u
    set depth(\( v_i \)) \rightarrow 1 + depth(u)
    DFS(\( v_i \))

\{ only if you want to keep the structure \}
Start with:

- mark $s$
- $\text{depth}(s) = 0$
- $\text{DFS}(s)$

**Time:** $O(1 \text{E})$

$O(V+E)$ if not connected

$\text{DFS}(u)$  // DFS starting at $u$

for every neighbor $v_i$ of $u$  // i.e. scan $\text{Adj}[u]$

  - if $v_i = t$, DONE
  - if $v_i$ is unmarked
    - mark $v_i$
    - set $\text{parent}(v_i) \rightarrow u$
    - set $\text{depth}(v_i) \rightarrow 1 + \text{depth}(u)$
    - $\text{DFS}(v_i)$

} only if you want to keep the structure
DFS on a non-connected graph $G$

For every vertex $v_i$ in $G$
if $v_i$ is unmarked
$\text{DFS}(v_i)$
DFS on a non-connected graph $G$

For every vertex $v_i$ in $G$
if $v_i$ is unmarked
$\text{DFS}(v_i)$

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
       $\text{DFS}(v_i)$

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
  - $\text{DFS}(v_i)$

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

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 is 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 \( \text{Adj}[v_1] \)
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?
3 before 1 in Adj[4]