Breadth-First Search

1. Starting from $s$, find vertices 1 edge from $s$.
2. Find vertices 2 edges from $s$.

- 3 edges
- 4 edges

- $O(n^2)$ storage if store radius pullout each node...

- $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \cdots \rightarrow n$

- Keep track of stats: unseen / seen / done

- Distance

- Parent

- Unseen

- Add next to queue as we see them
BFS(V,E,from)

for each vertex u in V
    color[u] ← UNSEEN
    d[u] ← infinity
    parent[u] ← NULL

color[from] ← PROCESSING
d[from] ← 0
parent[from] ← NULL
Q ← [from]

while not Q.isEmpty()
    u ← Q.dequeue()
    for each v adjacent to u
        if color[v] == UNSEEN
            parent[v] ← u
            d[v] ← d[u] + 1
            color[v] ← PROCESSING
            Q.enqueue(v)
    color[u] = DONE

O(n + m) total (another form of for each vertex v for adjacent list)
for each edge from v

O((n + m) total (using linked list or array w/ changing start index)
Depth-First Search

Depth-first search: Keep following edges from current vertex
Backtrack when no edges to unvisited vertices

finds cycles \( \text{(when sees edge to processing vertex)} \)

Cycles

when examining \( 6 \to 3 \), see that 3 is processing (recursive call not done)
\[ \text{cycle!} \]

Topological Sort — for a directed acyclic graph
orders verts so edges go in one direction
Topological Sort

for a directed acyclic graph
orders verts so edges go in one direction

Longest Path
DFS-VISIT(G, u)
   color[u] <- PROCESSING
   discovery[u] <- time++
   for each v adjacent to u
      if (color[v] == UNSEEN)
         pred[v] <- u
         DFS-VISIT(G, v)
   finish[u] <- time++
   color[u] <- DONE

DFS(G)
   for each u in G.V
      color[u] <- UNSEEN
   time <- 0
   for each u in G.V
      if color[u] = UNSEEN
         DFS-VISIT(G, u)

more useful in DFS-based algo than distance

to restart search
at new point when
completely stuck
(useful for some applications
of DFS, including topo sort)