Thought experiment. World social graph.
- Is it connected?
- If not, how big is largest connected component?
- Is there a path between you and Tom Brady? What about Theresa May?

How can you tell algorithmically?
Answer: graph traversal! (BFS/DFS)

Breadth-First Search: Layers
Explore outward from starting node \( s \).

Define layer \( L_i = \{ v \mid \text{distance } i \text{ from } s \} \).

- \( L_0 = \{ s \} \)
- \( L_1 = \text{nodes with edge to } L_0 \)
- \( L_2 = \text{nodes with an edge to } L_1 \text{ that don’t belong to } L_0 \text{ or } L_1 \)
- \( \ldots \)
- \( L_{i+1} = \text{nodes with an edge to } L_i \text{ that don’t belong to any earlier layer} \)

Observation: There is a path from \( s \) to \( t \) if and only if \( t \) appears in some layer.

Graph Traversal

Clicker

How many nodes are in layer 2, starting a BFS from MIT?

A) 4
B) 5
C) 6
D) None of the above
**BFS Implementation**

BFS(s):
- mark s as "discovered"
- L[0] ← {s}, i ← 0
- while L[i] is not empty do
  - L[i + 1] ← empty list
  - for all nodes v in L[i] do
    - for all neighbors w of v do
      - if w is not marked "discovered" then
        - mark w as "discovered"
        - put w in L[i]
    - end if
  - end for
  - i ← i + 1
- end while

Running time? How many times does each line execute?

---

**BFS Running Time**

BFS(s):
- mark s as "discovered"  ▶ 1
- L[0] ← {s}, i ← 0  ▶ 1
- while L[i] is not empty do  ▶ 1
  - L[i + 1] ← empty list
  - for all nodes v in L[i] do  ▶ n
    - for all neighbors w of v do  ▶ 2m
      - if w is not marked "discovered" then
        - mark w as "discovered"  ▶ n
      - put w in L[i]
    - end if
  - end for
  - i ← i + 1  ▶ n
- end while

Running time: $O(m + n)$. Hidden assumption: can iterate over neighbors of $v$ efficiently... OK pending data structure.

---

**BFS Tree**

We can use BFS to make a tree. (blue: "tree edges", dashed: "non-tree edges")

---

**BFS Tree**

Claim: let $T$ be the tree discovered by BFS on graph $G = (V, E)$, and let $(x, y)$ be any edge of $G$. Then the layer of $x$ and $y$ in $T$ differ by at most 1.

**Proof**
- Let $(x, y)$ be an edge
- Assume $x$ is discovered first and placed in $L_i$
- Then $y \in L_i$ for $j \geq i$
- When neighbors of $x$ are explored, $y$ is either already in $L_i$, or is discovered and added to $L_{i+1}$
Suppose in BFS that the nodes in each layer are explored in a different order (e.g. reverse). Which of the following are true?

A. The nodes that appear in each layer may change  
B. The BFS tree may change  
C. Both A and B  
D. Neither A nor B

Depth-First Search

Depth-first search (DFS): keep exploring from the most recently added node until you have to backtrack.

DFS: Recursive Implementation

DFS(u)
mark u as "explored"
for all edges (u, v) do
if v is not "explored" then
   call DFS(v) recursively
end if
end for

DFS Tree

Claim: Non-tree edges lead to (indirect) ancestors

DFS: Non-tree edges lead to ancestors

Claim: Let T be the tree discovered by DFS, and let (x, y) be an edge of G that is not in T. Then one of x or y is an ancestor of the other.

Proof:

- Let x be the first of the two nodes explored  
- Is y explored at beginning of DFS(x)? No.  
- At some point during DFS(x), we examine the edge (x, y). Is y explored then? Yes, otherwise we would put (x, y) in T  
- \( \Rightarrow y \) was explored during DFS(x)  
- \( \Rightarrow y \) is a descendant of x
Graph Representation: Adjacency Lists

Each node keeps list of neighbors

- Each edge stored twice
- Space? $\Theta(m + n)$
- Checking if $(u, v)$ is an edge? $O(\deg(u))$ time ($\deg$ = number of neighbors)

Generic Graph Traversal

Let $A$ = data structure of discovered nodes

Traverse($s$)

put $s$ in $A$

while $A$ is not empty do

take a node $v$ from $A$

if $v$ is not marked "explored" then

mark $v$ "explored"

for each edge $(v, w)$ incident to $v$ do

put $w$ in $A$ \> $w$ is discovered

end for

end if

end while

BFS: $A$ is a queue (FIFO) \> DFS: $A$ is a stack (LIFO)

Can a node be discovered (placed in $A$) multiple times? Yes.

For DFS, node is explored from parent that added it last (LIFO).

For BFS, can avoid by not adding discovered nodes.

Clicker Question 2

Put $s$ in $A$

while $A$ is not empty do

take a node $v$ from $A$

if $v$ is not marked "explored" then

mark $v$ "explored"

for each edge $(v, w)$ incident to $v$ do

put $w$ in $A$ \> $w$ is discovered

end for

end if

end while

What is the maximum number of times a node $w$ can be put in $A$?

- A: once
- B: $\deg(w)$ times
- C: $2 \cdot \deg(w)$ times
- D: $|V|$ times

Clicker Question 3

DFS($s$)

Mark $u$ as "explored"

for each edge $(u, v)$ do

if $v$ is not "explored" then

Call DFS($v$) recursively

end if

end for

Put $s$ in $A$

while $A$ is not empty do

Take a node $v$ from $A$

if $v$ is not "explored" then

Mark $v$ as "explored"

for each edge $(v, w)$ do

Put $w$ in $A$

end for

end if

end while

Suppose we have a tree with $n$ nodes, height $h$ and degree $d$.

Compare recursive and non-recursive DFS in terms of memory used for the stack

- A: recursive: $\Theta(hd)$, non-recursive: $\Theta(h)$
- B: recursive: $\Theta(h)$, non-recursive: $\Theta(d)$
- C: recursive: $\Theta(n)$, non-recursive: $\Theta(hd)$
- D: recursive: $\Theta(h)$, non-recursive: $\Theta(hd)$

Exploring all Connected Components

How to explore entire graph even if it is disconnected?

while there is some unexplored node $s$ do

Traverse($s$) \> Run BFS/DFS starting from $s$.

Extract connected component containing $s$

end while

Running time? Does it change?

Naive: $O(m + n)$ per component $\Rightarrow O(c(m + n))$ if $c$ components.

Better: Search on component $C$ only works on nodes/edges in $C$

- Time for component $C$: $O(#edges in C + #nodes in C)$
- Total time still $O(m + n)$

Usually OK to assume graph is connected.

State if you are doing so and why it does not trivialize the problem.
Summary

- Graph traversal by BFS/DFS: basic algorithmic primitive used in many other algorithms
  - Is there a path from u to v?
  - Find all connected components
  - Produce trees with different properties, sometimes useful in algorithms
- $\Theta(m + n)$ time
- Different versions of general exploration strategy