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 $L_i = \{t \mid i = \text{distance from } s \text{ to } t\}$.

- $L_0 = \{s\}$
- $L_1 = \{t \mid s \text{ has edge to } t \}$
- $L_2 = \{t \mid t \text{ has edge to a node in } L_1 \}$
- $\ldots$
- $L_{i+1} = \{t \mid t \text{ has edge to a node in } L_i \}$

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

**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"
  \( L[0] \leftarrow \{s\}, \; i \leftarrow 0 \)  \( \triangleright \) Discover s
  while \( L[i] \) is not empty do
    \( L[i + 1] \leftarrow \) 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 \( L[i + 1] \)
        \end if
      \end for
    \end for
    \( i \leftarrow i + 1 \)
  \end while

Running time? How many times does each line execute?

BFS Running Time

BFS(s):
  mark s as "discovered"
  \( L[0] \leftarrow \{s\}, \; i \leftarrow 0 \)  \( \triangleright 1 \)
  while \( L[i] \) is not empty do
    \( L[i + 1] \leftarrow \) 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
    \end for
    \( i \leftarrow i + 1 \)
  \end while

Running time: \( O(|V| + |E|) \). 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”)

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 \( y \in L_j \) for \( j \geq i \)
- When neighbors of \( x \) are explored, \( y \) is either already in \( L_i \) or \( L_{i+1} \), or is discovered and added to \( L_{i+1} \)

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 \( y \in L_j \) for \( j \geq i \)
- When neighbors of \( x \) are explored, \( y \) is either already in \( L_i \) or \( L_{i+1} \), or is discovered and added to \( L_{i+1} \)

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 \( y \in L_j \) for \( j \geq i \)
- When neighbors of \( x \) are explored, \( y \) is either already in \( L_i \) or \( L_{i+1} \), or is discovered and added to \( L_{i+1} \)

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 \( y \in L_j \) for \( j \geq i \)
- When neighbors of \( x \) are explored, \( y \) is either already in \( L_i \) or \( L_{i+1} \), 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: Running Time**

How to analyze if algorithm is recursive? Same: count executions of each line, including recursive call

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

Running time: \( O(m + n) \) same complexity as BFS
Same assumptions: can traverse neighbor list in time proportional to node degree

**DFS Tree**

\( T \leftarrow \text{empty} \)

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

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 \( T \)
- \( y \) was explored during DFS(\( x \))
- \( y \) is a descendant of \( x \)
Graph Representation: Adjacency Lists

- Each node keeps list of neighbors
- Space? \( \Theta(m + n) \)
- Checking if \((u, v)\) is an edge? \( O(\text{degree}(u)) \) time (degree = number of neighbors)

Generic Graph Traversal

Let \( A \) = data structure of discovered nodes

1. Traverse(s)
   1. put \( s \) in \( A \)
   2. while \( A \) is not empty do
      1. take a node \( v \) from \( A \)
      2. if \( v \) is not marked "explored" then
         1. mark \( v \) "explored"
         2. for each edge \((v, w)\) incident to \( v \) do
            1. put \( w \) in \( A \)  \( \triangleright \) \( w \) 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
  1. Take a node \( v \) from \( A \)
  2. if \( v \) is not "explored" then
     1. Mark \( v \) "explored"
     2. for each edge \((v, w)\) do
        1. Put \( w \) in \( A \)  \( \triangleright \) \( w \) 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: \( \text{degree}(w) \) times
- C: \( 2 \cdot \text{degree}(w) \) times
- D: \( |V| \) times

Clicker Question 3

- DFS(s)
  1. Mark \( s \) as "explored"
  2. for each edge \((u, v)\) do
     1. if \( v \) is not "explored" then
        1. Call DFS(v) recursively
   end if
end for

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?
  1. while there is some unexplored node \( s \) do
     1. Traverse(s)  \( \triangleright \) Run BFS/DFS starting from \( s \).
     2. 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 \( C \)
  1. Time for component \( C \): \( O(|\text{edges in } C| + |\text{nodes in } C|) \)
  2. 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