### BFS So Far

Explore outward from $s$ by distance $\rightarrow$ layers

Layer $L_i =$ nodes at distance $i$

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

### BFS Implementation

BFS($s$):

1. mark $s$ as "discovered"
2. $L[0] \leftarrow \{s\}, i \leftarrow 0$
3. 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 $L[i + 1]$
   - $i \leftarrow i + 1$

Running time? How many times does each line execute? (For now, assume graph is connected)

### BFS Running Time

BFS($s$):

1. mark $s$ as "discovered"  \(\triangleright 1\)
2. $L[0] \leftarrow \{s\}, i \leftarrow 0$  \(\triangleright 1\)
3. while $L[i]$ is not empty do  \(\triangleright \leq n\)
   - $L[i + 1] \leftarrow$ empty list  \(\triangleright \leq n\)
   - for all nodes $v$ in $L[i]$ do  \(\triangleright n\)
     - for all neighbors $w$ of $v$ do  \(\triangleright 2m\)
       - if $w$ is not marked "discovered" then  \(\triangleright 2m\)
         - mark $w$ as "discovered"  \(\triangleright n\)
         - put $w$ in $L[i + 1]$  \(\triangleright n\)
   - $i \leftarrow i + 1$  \(\triangleright \leq n\)

Running time: $\Theta(m + n)$

- Another way to think about it: "touch each node and edge" a constant number of times
- Hidden assumption: can iterate over neighbors of $v$ efficiently...
Clicker

Let \( q = \sum_{v \in V} \text{degree}(v) \) (this is the sum of degrees of all nodes in the graph).
Which one of the following is false?

A. \( q \) is twice the number of edges
B. \( q \) is \( n \) times the average degree
C. \( q \) is \( \Theta(m + n) \) if \( m \geq n \)
D. None of the above

Graph Representation: Adjacency Lists

Each node keeps list of neighbors

- Each edge stored twice
- Space? \( \Theta(m + n) \)
- Time to check if \((u, v)\) is an edge? \( O(\text{degree}(u)) \)
  (degree = number of neighbors)
- Time to iterate over all neighbors of \( v \)? \( O(\text{degree}(u)) \)

BFS Tree

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

\[
\text{BFS}(s): \\
\text{mark } s \text{ as "discovered"} \\
L[0] \leftarrow \{s\}, \quad i \leftarrow 0 \\
T \leftarrow \text{empty} \\
\text{while } L[i] \text{ is not empty do} \\
\quad L[i + 1] \leftarrow \text{empty list} \\
\quad \text{for all nodes } v \text{ in } L[i] \text{ do} \\
\quad \quad \text{if } w \text{ is not marked "discovered" then} \\
\quad \quad \quad \text{mark } w \text{ as "discovered"} \\
\quad \quad \quad \text{put } w \text{ in } L[i + 1] \\
\quad \quad \text{put } (v, w) \text{ in } T \\
\quad i \leftarrow i + 1
\]

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 \( y \in L_j \) 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

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

Running time: O(m + n) same as BFS

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
- ⇒ y was explored during DFS(x)
- ⇒ y is a descendant of x
Generic Traversal Implementations

Generic approach: maintain set of explored nodes and discovered nodes

- Explored = have seen this node and explored its outgoing edges
- Discovered = the “frontier”. Have seen the node, but not explored its outgoing edges.

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 $A$  \(\triangleright w \text{ is discovered} \)

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

Clicker

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 $A$ \(\triangleright w \text{ is discovered} \)

Suppose we run this traversal code and every node is marked explored before it terminates. Which of the following is false?

A. Every node is marked “explored” exactly once.
B. A single node could be put into $A$ more than once.
C. If $w \neq s$, the number of times that node $w$ is put into $A$ is degree($w$).
D. It’s possible that there exist nodes $x$ and $y$ with no path from $x$ to $y$.

Exploring all Connected Components

How to explore entire graph even if it is disconnected?

while there is some unexplored node $s$ do

Traverse($s$)

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

Extract connected component containing $s$

Running time? Still $O(m + n)$

- Traversal of each component takes time proportional to the numbers of nodes + edges in that component

Advice: 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 $v$ 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