Objectives

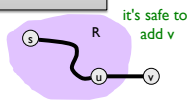
- Graph Searching: BFS and DFS
  - Analysis and implementation

Review: Graphs

- What do we know about graphs?
- How will we typically represent graphs?

Review: A General Algorithm for Finding Connected Component

```
R will consist of nodes to which s has a path
R = {s}
while there is an edge (u,v) where u∈R and v∉R
      add v to R
```



- *R* will be the **connected component** containing s
- Algorithm is underspecified
  - BFS and DFS say how to consider edges
- Theorem. Upon termination, R is the connected component containing s

Connected Component: BFS

- Find all nodes ***reachable*** from *s*

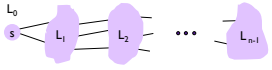
In general….

```
R will consist of nodes to which s has a path
R = {s}
while there is an edge (u,v) where u∈R and v∉R
      add v to R
```

In what order does BFS consider edges?

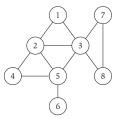
Breadth-First Search

- **Intuition**. Explore outward from *s* in all possible directions (edges), adding nodes one "layer" at a time
- **Algorithm**
  - $L_0 = \{ s \}$
  - L_1 = all neighbors of L_0
  - L_2 = all nodes that have an edge to a node in L_1 and do not belong to L_0 or L_1
  - L_{i+1} = all nodes that have an edge to a node in L_i and do not belong to an earlier layer



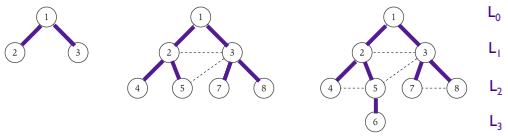
Run BFS on This Graph

s = 1



Example of Breadth-First Search

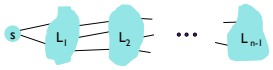
s = 1



Creates a tree
-- (dashed line) is a node in the graph that is not in the tree

Breadth-First Search

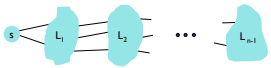
- **Theorem.**
  For each *i*, L_i consists of all nodes at distance exactly *i* from *s*.
  There is a path from s to t iff t appears in some layer.



- What does this theorem mean?
  - What is the significance of *i*?

Breadth-First Search

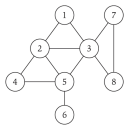
- **Theorem.** For each *i*, L_i consists of all nodes at distance exactly *i* from *s*. There is a path from *s* to *t* iff *t* appears in some layer.
  - Shortest path to *t* from *s*, is the *i* from L_i
  - All nodes ***reachable*** from *s* are in L_1, L_2, …, L_{n-1}



Breadth-First Search

- **Property**. Let T be a BFS tree of G = (V, E), and let (x, y) be an edge of G. Then the level of x and y *differ* by *at most* 1.

G:



If x is in L_i,
then y must be in ???

Connected Component: DFS

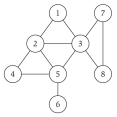
- Find all nodes ***reachable*** from *s*

In general….

```
R will consist of nodes to which s has a path
R = {s}
while there is an edge (u,v) where u∈R and v∉R
      add v to R
```

In what order does DFS consider edges?

Depth-First Search

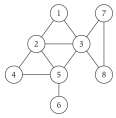


- Need to keep track of where you've been
- When reach a "dead-end" (already explored all neighbors), backtrack to node with unexplored neighbor
- Algorithm:

```
DFS(u):
   Mark u as "Explored" and add u to R
   For each edge (u, v) incident to u
      If v is not marked "Explored" then
            DFS(v)
```

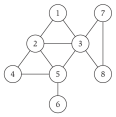
Depth-First Search

- How does DFS work on this graph?
  - Starting from node 1



DFS vs BFS

- Compare the resulting trees



DFS Analysis

- Let T be a depth-first search tree, let *x* and *y* be nodes in T, and let (*x*, *y*) be an edge of G that is not an edge of T. Then one of *x* or *y* is an ancestor of the other in T.

"analogous to" BFS: connected nodes are at most one layer apart

DFS Analysis

- Let T be a depth-first search tree, let *x* and *y* be nodes in T, and let (*x*, *y*) be an edge of G that is not an edge of T. Then one of *x* or *y* is an ancestor of the other in T.
- Proof.
  - Suppose that x-y is an edge in G but not in T. (From problem statement)
  - WLOG, assume that DFS reaches x before y
  - When edge x-y is considered in the DFS algorithm, we don't add it to T (from problem statement), which means that y must have been explored.
  - But, since we reached x first, y had to be discovered between invocation and end of the recursive call DFS(x)
    - i.e., y is a descendent of x

Analysis of Connected Components

- For any two nodes *s* and *t* in a graph, their connected components are either identical or disjoint
- Proof?

Analysis of Connected Components

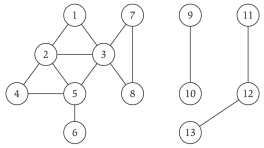
- For any two nodes *s* and *t* in a graph, their connected components are either identical or disjoint
- Proof sketch:

  (i) There is a path between *s* and *t* → same set of connected components

  (ii) There is no path between *s* and *t* → disjoint set of connected components

Set of All Connected Components

- How can we find set of ***all*** connected components of graph?



Set of All Connected Components

- How can we find set of all connected components of graph?

```
R* = set of connected components
while there is a node that does not belong to R*

    select s not in R*

    R = {s}

    while there is an edge (u,v) where u∈R and v∉R
       add v to R


    Add R to R*
```

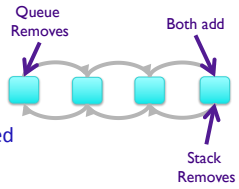
Find s's connected component

IMPLEMENTATION & ANALYSIS

Queues and Stacks

- How are queues and stacks similar?
- How are queues and stacks different?

Queues and Stacks



- Both: doubly linked list
  - Always take first on list
  - Difference in where extracted
  - Have first and last pointers
  - Done in constant time
- Queue: FIFO
  - First in, first out
- Stack: LIFO
  - Last in, first out

Looking Ahead

- Reading Chapter 2.5, 3-3.2 - tonight
- Problem Set 3 due before class on Wednesday
- Midterm given on Wednesday
  - Take home
  - Due next Wednesday at 5 p.m.
  - Open notes (yours, journal), book, and lectures
  - Open me
  - Closed to everything else