Objectives

- Graph Traversal
- BFS & DFS Implementations, Analysis

Notes on Assignments

- Designing algorithms
  - Be as descriptive as possible, provide intuition
  - Explain running time
    - Match prescribed running time
    - Or what you think the running time is
- Wiki
  - Say something about how readable/interesting the section was on scale of 1 to 10

Review: Comparing BFS vs DFS

- What do they do?
- How are their outcomes different?
- When would we want to use one over the other?

Review: Comparing BFS vs DFS

- What do they do?
  - Techniques for finding connected components
    - Create a tree of connected components
  - Other uses as well
- How are their outcomes different?
  - BFS: shortest path; bushy tree
  - DFS: spindly tree
- When would we want to use one over the other?
  - DFS: what you'd do in a maze (can't split)

Connected Component

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

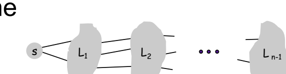
In general....

```
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
```

- Theorem. Upon termination, R is the connected component containing s
  - BFS = explore in order of distance from s
  - DFS = explore until hit "deadend"

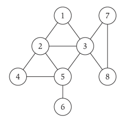
Breadth-First Search

- **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 do not belong to L_0 or L_1 and that have an edge to a node in L_1
  - L_{i+1} = all nodes that do not belong to an earlier layer and that have an edge to a node in L_i

Depth-First Search

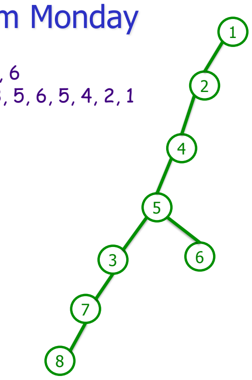


- Need to keep track of where you've been
- When reach a "dead-end" (already explored all neighbors), backtrack to node with unexplored neighbor
- Algorithm:

```
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)
```


Our DFS Tree from Monday

Explored: 1, 2, 4, 5, 3, 7, 8, 6
Now: 1, 2, 4, 5, 3, 7, 8, 7, 3, 5, 6, 5, 4, 2, 1
R: 1, 2, 4, 5, 7, 8, 6



DFS Analysis

- Let T be a depth-first search tree, let *x* and *y* by 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.

DFS Analysis

- Let T be a depth-first search tree, let *x* and *y* by 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.
- 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 the set of ***all*** connected components of graph?

Set of All Connected Components

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

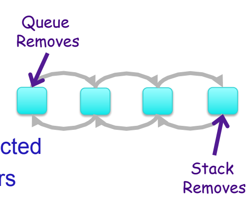
```
R* = set of connected components
While there is a node that does not belong to R*

    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*
```


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, last out

Described differently in book
- Inserted differently
- Extracted at same place

Implementing BFS

- Graph: Adjacency list
- Discovered array
- Maintain layers in separate lists, L[i]

Implementing BFS

- Graph: Adjacency list
- Discovered array
- Maintain layers in separate lists, L[i]

```
BFS(s):
    Discovered[v] = false, for all v
    Discovered[s] = true
    L[0] = {s}
    layer counter i = 0
    BFS tree T = {}
    while L[i] != {}
        L[i+1] = {}
        For each node u ∈ L[i]
            Consider each edge (u,v) incident to u
            if Discovered[v] == false then
                Discovered[v] = true
                Add edge (u, v) to tree T
                Add v to the list L[i + 1]
        i+=1
```

What does this stopping condition mean?

L[i] as a queue or stack?

Analysis

```
BFS(s):
    Discovered[v] = false, for all v
    Discovered[s] = true
    L[0] = {s}
    layer counter i = 0
    BFS tree T = {}
    while L[i] != {}
        L[i+1] = {}
        For each node u ∈ L[i]
            Consider each edge (u,v) incident to u
            if Discovered[v] == false then
                Discovered[v] = true
                Add edge (u, v) to tree T
                Add v to the list L[i + 1]
        i+=1
```

L[i] as a queue or stack?

- Doesn't matter because algorithm can consider nodes in any order

Analysis

n

$O(n^2)$

At most n

At most n-1

```
BFS(s):
    Discovered[v] = false, for all v
    Discovered[s] = true
    L[0] = {s}
    layer counter i = 0
    BFS tree T = {}
    while L[i] != {}
        L[i+1] = {}
        For each node u ∈ L[i]
            Consider each edge (u,v) incident to u
            if Discovered[v] == false then
                Discovered[v] = true
                Add edge (u, v) to tree T
                Add v to the list L[i + 1]
        i+=1
```


Analysis: Tighter Bound

n

At most n

O(deg(u))

```
BFS(s):
    Discovered[v] = false, for all v
    Discovered[s] = true
    L[0] = {s}
    layer counter i = 0
    BFS tree T = {}
    while L[i] != {}
        L[i+1] = {}
        For each node u ∈ L[i]
            Consider each edge (u,v) incident to u
            if Discovered[v] == false then
                Discovered[v] = true
                Add edge (u, v) to tree T
                Add v to the list L[i + 1]
        i+=1
```

$\Sigma_{u \in V} \deg(u) = 2m$

$\rightarrow O(n+m)$

Implementing DFS

- Defined iteratively rather than recursively
  - Analogous to BFS

Implementing DFS

- Keep nodes to be processed in a *stack*

```
DFS(s):
    Initialize S to be a stack with one element s
    Explored[v] = false, for all v
    Parent[v] = 0, for all v
    DFS tree T = {}
    while S != {}
        Take a node u from S
        If Explored[u] = false
            Explored[u] = true
            Add edge (u, parent[u]) to T (if u ≠ s)
            For each edge (u, v) incident to u
                Add v to the stack S
                Parent[v] = u
```

Assignments

- Continue reading Chapter 3
  - Post summaries on Wiki
- Problem Set 2 due Friday