Objectives

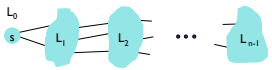
- Wrapping up implementing BFS and DFS
- Graph Application: Bipartite Graphs
- Directed Graphs

Review: Comparing BFS vs DFS

- What do they do?
- How are their outcomes different?

Review: 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



Implementing BFS

- What do we need as input?
- What do we need to model?
  - How will we model that?

Implementing BFS

- Input: Graph as an adjacency list
- Discovered array
- Maintain layers in separate lists, L[i]

Implementing BFS

- Graph: Adjacency list
- Discovered array
- Maintain layers L[i]

What does this stopping condition mean?

L[i] representation?

```
BFS(s, G):
    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

```
BFS(s, G):
    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] representation? List, queue, or stack
  - Doesn't matter because algorithm can consider nodes in any order

What is the running time?

Analysis

```
BFS(s, G):
    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
```

n; At most n; At most n−1; $O(n^3)$

Analysis: Tighter Bound

```
BFS(s, G):
    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
```

n; At most n; At most n−1; $O(n^2)$

Because we're going to look at each node at most once

Analysis: Even Tighter Bound

```
BFS(s, G):
    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
```

n; At most n; O(deg(u)); $\Sigma_{u \in V} \deg(u) = 2m$

→ O(n+m)

Implementing DFS

- What do we need as input?
- What do we need to model?
  - How will we model that?
  - Pseudo code

```
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)
```


Implementing DFS

- Keep nodes to be processed in a *stack*

```
DFS(s, G):
    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
```

What is the runtime?
How many times is a node added/removed from the stack?

Analyzing DFS

O(n+m)

```
DFS(s, G):
   Initialize S to be a stack with one element s
   Explored[v] = false, for all v
   Parent[v] = 0, for all v                    } O(n)
   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)
deg(u)   for each edge (u, v) incident to u
            Add v to the stack S
            Parent[v] = u
```

A node is added/removed from the stack 2* deg(u)
All nodes are added 2m = O(m) times

Analyzing Finding All Connected Components

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

```
R* = set of connected components (a set of sets)
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*
```

But the inner loop is O(m+n)!
How can this RT be possible?

Running time: O(m+n)

Set of All Connected Components

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

```
R* = set of connected components (a set of sets)
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*
```

Imprecision in the running time of inner loop: O(m+n)

But that's m and n of the *connected* component, let's say m_i and n_i .
$\Sigma_i O(m_i + n_i) = O(m+n)$

Where *i* is the subscript of the connected component

Looking Ahead

- Exam 1 – due next Wednesday
  - Closed to most
    - Open book, notes, journals, me
  - Work period Monday
- No wiki for Monday