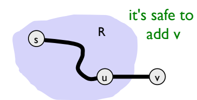
Objectives

- BFS & DFS Implementations, Analysis
- Graph Application: Bipartiteness

Review: Finding Connected Components

```
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
```



DFS and BFS say what order we look at the edges.

Review

- Why would we want to find all the connected components in a graph?
  - applications
- 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?
  - BFS: Shortest path
  - DFS: what you'd do in a maze (can't split)

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's 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 a graph?

Set of All Connected Components

- How can we find set of ***all*** connected components of a 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*
```

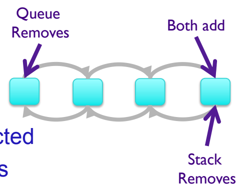

IMPLEMENTATION & ANALYSIS

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



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 DFS

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

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] representation?

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] representation? List, queue, or stack
  - Doesn't matter because algorithm can consider nodes in any order

What is the running time?

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
```

n

At most n

At most n-1

At most n-1

$O(n^3)$

Analysis: Tighter Bound

```
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
```

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):
  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)

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

Problem Set #1

- $\sqrt{2}n < n + 10$
- $n^2 \log n < n^{2.5}$
  - $\log n < n^{.5}$ (divide by n^2)
  - $\log \log n < .5 * \log n$ (take log of each)
- Similar to solved problem in Chapter 2

Reminders

- Friday: Problem Set 2 due
  - See HeapBottomUp