Objectives

Data structures: Graphs

From the *Office*

It's my own fault for using PowerPoint. PowerPoint is boring. People learn in a lot of different ways.

-- Dwight

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)

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 will consist of nodes to which s has a path
    Initially R = {s}
    While there is an edge (u,v) where u ∈ R and v ∉ R
      Add v to R
    Endwhile

     Add R to R*
```

Running time?

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 will consist of nodes to which s has a path
    Initially R = {s}
    While there is an edge (u,v) where u ∈ R and v ∉ R
      Add v to R
    Endwhile

     Add R to R*
```

Running time: O(m+n)

But the "inner" loop was O(m+n)!
How can this be?

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 will consist of nodes to which s has a path
    Initially R = {s}
    While there is an edge (u, v) where u ∈ R and v ∉ R
      Add v to R
    Endwhile
    ------------------------------------------------
    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

So…

$\Sigma_i\ O(m_i + n_i) = O(m+n)$

Where i is the subscript of the connected component

TESTING BIPARTITENESS

Bipartite Graphs

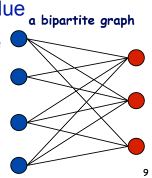
Def. An undirected graph G = (V, E) is bipartite if the nodes can be colored red or blue such that every edge has one red and one blue end

- Generally: vertices divided into sets X and Y

Applications:

- Stable marriage: men = red, women = blue
- Scheduling: machines = red, jobs = blue

a bipartite graph

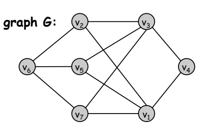


Testing Bipartiteness

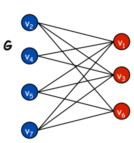
Given a graph G, is it bipartite?

- Many graph problems become:
  - easier if underlying graph is bipartite (matching)
  - tractable if underlying graph is bipartite (independent set)
- Before designing an algorithm, need to understand structure of bipartite graphs

a bipartite graph G:



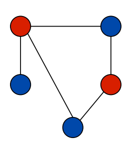
another drawing of G



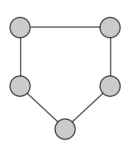
An Obstruction to Bipartiteness

Lemma. If a graph G is bipartite, it cannot contain an odd length cycle.

Pf. Not possible to 2-color the odd cycle, let alone G.



bipartite
(2-colorable)



not bipartite
(not 2-colorable)

If find an odd cycle, graph is NOT bipartite

How Can We Determine Bipartite Graphs?

Given a connected graph Why connected?

Color one node red

- Doesn't matter which color (Why?)

What should we do next?

How will we know that we're finished?

What does this process sound like?

How Can We Determine Bipartite Graphs?

Given a connected graph

Color one node red

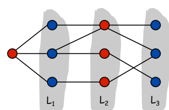
- Doesn't matter which color (Why?)

What should we do next?

How will we know that we're finished?

What does this process sound like?

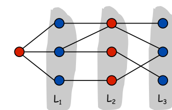
BFS: alternating colors, layers



Implementing Algorithm

Modify BFS to have a `Color` array

- When add v to list L[i+1]
- Color[v] = red if i+1 is even
- Color[v] = blue if i+1 is odd



Bipartite Graphs

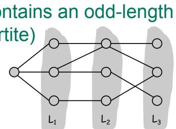
Lemma. Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds:

(i) No edge of G joins two nodes of the same layer

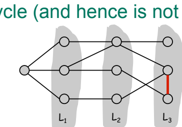
- G is bipartite

(ii) An edge of G joins two nodes of the same layer

- G contains an odd-length cycle (and hence is not bipartite)



Case (i)



Case (ii)

Bipartite Graphs

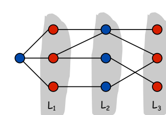
Lemma. Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds:

(i) No edge of G joins two nodes of the same layer

- G is bipartite

Pf. (i)

- Suppose no edge joins two nodes in the same layer
- Implies all edges join nodes on adjacent level
- Bipartition: red = nodes on odd levels, blue = nodes on even levels



Case (i)

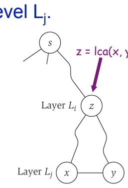
Bipartite Graphs

Lemma. Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds:

(ii) An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite)

Pf. (ii)

- Suppose (x, y) is an edge with x, y in same level L_j.
- Let z = lca(x, y) = lowest common ancestor
- Let L_i be level containing z
- Consider cycle that takes edge from x to y, then path from y to z, then path from z to x



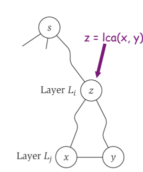
Bipartite Graphs

Lemma. Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds:

(ii) An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite)

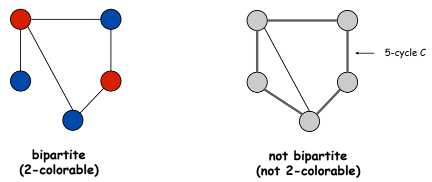
Pf. (ii)

- Suppose (x, y) is an edge with x, y in same level L_j
- Let z = lca(x, y) = lowest common ancestor
- Let L_i be level containing z
- Consider cycle that takes edge from x to y, then path from y to z, then path from z to x
- Its length is $\underbrace{1}_{(x,\,y)} + \underbrace{(j-i)}_{\text{path from y to z}} + \underbrace{(j-i)}_{\text{path from z to x}}$, which is odd



Obstruction to Bipartiteness

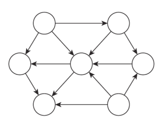
Corollary. A graph G is bipartite iff it contains no odd length cycle.



CONNECTIVITY IN DIRECTED GRAPHS

Directed Graphs G = (V, E)

Edge (u, v) goes from node u to node v

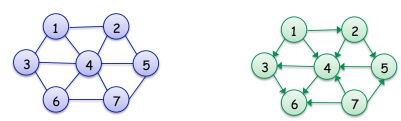


Ex. Web graph - hyperlink points from one web page to another

- Directedness of graph is crucial
- Modern web search engines exploit hyperlink structure to rank web pages by importance

Graph Search

How does *reachability* change with directed graphs?



Example: Web crawler. Start from web page s. Find all web pages linked from s, either directly or indirectly.

Representing Directed Graphs

For each node, keep track

- Out edges (where links go)
- In edges (from where links come in)

Could just keep out edges

- Get in edges with increased computation/time
- Useful to have both in and out edges

Graph Search

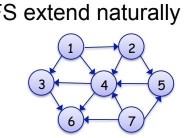
Directed reachability. Given a node s, find all nodes reachable from s.

Directed s-t shortest path problem. Given two nodes s and t, what is the length of the shortest path between s and t?

- Not necessarily the same as t-s shortest path

Graph search. BFS and DFS extend naturally to directed graphs

- Trace through out edges
- Run in O(m+n) time



Problem

Rather than paths from *s* to other nodes, find all nodes with paths to *s*

Problem/Solution

Problem. Rather than paths from *s* to other nodes, find all nodes with paths to *s*

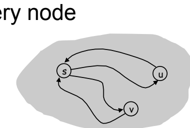
Solution. Run BFS on *in edges* instead of out edges

Strong Connectivity

Def. Node *u* and *v* are *mutually reachable* if there is a path from *u* to *v* and also a path from *v* to *u*

Def. A graph is *strongly connected* if every pair of nodes is mutually reachable

Lemma. Let *s* be any node. G is strongly connected iff every node is reachable from *s* and *s* is reachable from every node



Strong Connectivity

If *u* and *v* are mutually reachable and *v* and *w* are mutually reachable, then *u* and *w* are mutually reachable

Strong Connectivity

If *u* and *v* are mutually reachable and *v* and *w* are mutually reachable, then *u* and *w* are mutually reachable.

Proof. We need to show that there is a path from *u* to *w* and from *w* to *u*.

- By defn of mutually reachable, there is a path from *u* to *v*, a path from *v* to *u*, a path from *v* to *w*, and a path from *w* to *v*
- Take path *u*→*v* and then from *v* → w
  - Path from *u*→*w*
- Similarly for *w*→*u*

Strong Connectivity

Def. A graph is strongly connected if every pair of nodes is mutually reachable

Lemma. Let *s* be any node. G is **strongly connected** iff every node is reachable from *s* and *s* is reachable from every node.

- 1st prove $\Rightarrow$
- 2nd prove $\Leftarrow$
  - for any nodes *u* and *v*, is there a path u→v and v→u ?

Strong Connectivity

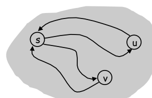
Def. A graph is *strongly connected* if every pair of nodes is mutually reachable

Lemma. Let *s* be any node. G is strongly connected iff every node is reachable from *s*, and *s* is reachable from every node.

Pf. $\Rightarrow$ Follows from definition of strongly connected

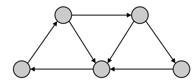
Pf. $\Leftarrow$ For any nodes *u* and *v*, make path *u*→*v* and *v*→*u*

- *u*→*v* : concatenating *u*→*s* with *s*→*v*
- *v* →*u*: concatenate *v*→*s* with *s*→*u*

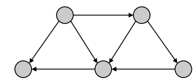


Strong Connectivity Problem

Determine if G is strongly connected in O(m + n) time



strongly connected



not strongly connected

Can we leverage any algorithms we know have O(m+n) time?

Strong Connectivity: Algorithm

Theorem. Can determine if G is strongly connected in O(m + n) time.

Pf.

- Pick any node *s*
- Run BFS from *s* in G
- Run BFS from *s* in G^{rev}

reverse orientation of every edge in G
Or, the BFS using the in edges

- Return true iff all nodes reached in both BFS executions
- Correctness follows immediately from previous lemma
  - All reachable from one node, s is reached by all

Strong Components

For any two nodes *s* and *t* in a directed graph, their strong components are either identical or disjoint

Consider a node in common…

Strong Components

For any two nodes *s* and *t* in a directed graph, their strong components are either identical or disjoint

Proof.

- Consider *v* in both strong components
  - s→ v; v → s; v→t; t→v ➔ t→s, s→t (mutually reachable)
  - As soon as there is one common node, then have identical strong components

DAGS AND TOPOLOGICAL ORDERING

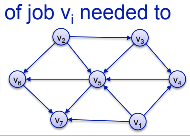
Directed Acyclic Graphs

Def. A DAG is a directed graph that contains *no directed cycles*.

Example. Precedence constraints: edge (v_i, v_j) means v_i must precede v_j

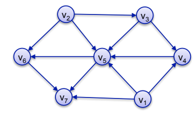
- Course prerequisite graph: course v_i must be taken before v_j
- Compilation: module v_i must be compiled before v_j
- Pipeline of computing jobs: output of job v_i needed to determine input of job v_j

a DAG:



Directed Acyclic Graphs

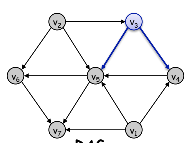
Given a set of tasks with dependencies, what is a valid order in which the tasks could be performed?



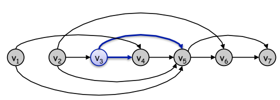
Directed Acyclic Graphs

Given a set of tasks with dependencies, what is a valid order in which the tasks could be performed?

Def. A ***topological order*** of a directed graph G = (V, E) is an ordering of its nodes as $v_1, v_2, \ldots, v_n$ so that for every edge (v_i, v_j) we have $i < j$.



a DAG

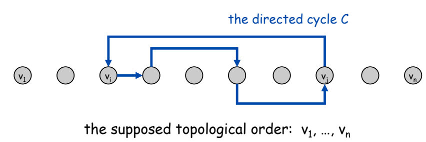


a topological ordering
All edges point "forward"

Directed Acyclic Graphs

Lemma. If G has a topological order, then G is a DAG.

Proof: Try to show that G has a cycle



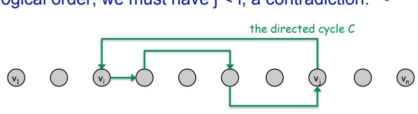
Why isn't this valid?

Directed Acyclic Graphs

Lemma. If G has a topological order, then G is a DAG.

Pf. (by contradiction)

- Suppose that G has a topological order $v_1, \ldots, v_n$ and that G also has a directed cycle C.
- Let v_i be the lowest-indexed node in C, and let v_j be the node on C just before v_i; thus (v_j, v_i) is an edge
- By our choice of i (lowest-indexed node), $i < j$
- On the other hand, since (v_j, v_i) is an edge and $v_1, \ldots, v_n$ is a topological order, we must have $j < i$, a contradiction. ▪



Directed Acyclic Graphs

Does every DAG have a topological ordering?

- If so, how do we compute one?

Directed Acyclic Graphs

Does every DAG have a topological ordering?

- If so, how do we compute one?

What would we need to be able to create a topological ordering?

- What are some characteristics of this graph?