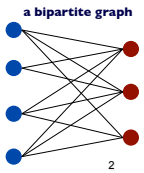
Objectives

- Graph Application: Bipartite Graphs
- Directed Graphs

Review: Bipartite Graphs

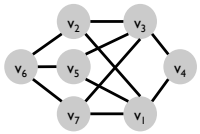
- Def. An undirected graph G = (V, E) is ***bipartite*** if the nodes can be colored red or blue such that every edge has one red and one blue end
  - Generally: vertices divided into sets X and Y
- Applications:
  - Stable marriage:
    - men = red, women = blue
  - Scheduling:
    - machines = red, jobs = blue



Where we left off: How Can We Determine if a Graph is Bipartite?

- Given a connected graph
  1. Color one node red
     - Doesn't matter which color (Why?)
  - What should we do next?

Why connected?

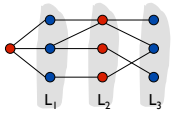


- How will we know when we're finished?
- What does this process sound like?

How Can We Determine if a Graph is Bipartite?

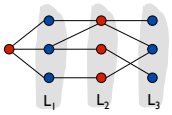
- Given a connected graph
  - Color one node red
    - Doesn't matter which color (Why?)
  - What should we do next?
- How will we know that we're finished?
- What does this process sound like?
  - BFS: alternating colors, layers

How can we implement the algorithm?



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



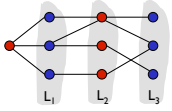
What is the running time of this algorithm? **O(n+m)**

Marks a change in how we think about algorithms
Starting to apply known algorithms to solve new problems.

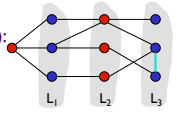
Analyzing Algorithm's Correctness

- Lemma. Let G be a connected graph, and let L_0, …, 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):



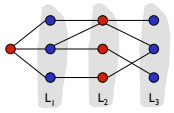
Case (ii):



Analyzing Algorithm's Correctness

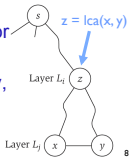
- Lemma. Let G be a connected graph, and let L_0, …, L_k be the layers produced by BFS starting at node *s*. Exactly one of the following holds:
  - (i) No edge of G joins two nodes of the same layer
    - ⇒ G is bipartite
- Pf. (i)
  - Suppose no edge joins two nodes in the same layer
  - Implies all edges join nodes on adjacent level
  - Bipartition: red = nodes on odd levels, blue = nodes on even levels

Case **(i)**



Analyzing Algorithm's Correctness

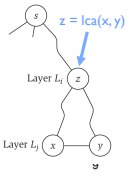
- Lemma. Let G be a connected graph, and let L_0, …, L_k be the layers produced by BFS starting at node *s*. Exactly one of the following holds:
  - (ii) An edge of G joins two nodes of the same layer → G contains an odd-length cycle and hence is not bipartite
- Pf. (ii)
  - Suppose (x, y) is an edge 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 y →z, then path from z → x



Analyzing Algorithm's Correctness

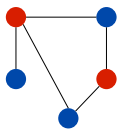
- Lemma. Let G be a connected graph, and let L_0, …, L_k be the layers produced by BFS starting at node *s*. Exactly one of the following holds:
  - (ii) An edge of G joins two nodes of the same layer → G contains an odd-length cycle and hence is not bipartite
- Pf. (ii)
  - Suppose (x, y) is an edge 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 y → z, then path z → x
  - Its length is 1 + (j-i) + (j-i), which is odd

(x, y) path from y to z path from z to x

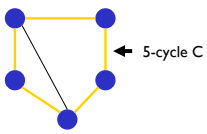


Obstruction to Bipartiteness

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



bipartite (2-colorable)

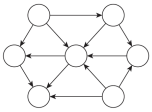


not bipartite (not 2-colorable)

DIRECTED GRAPHS

Directed Graphs G = (V, E)

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



- Example: 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

Representing Directed Graphs

- For each node, keep track of
  - Out edges (where links go)
  - In edges (from where links come in)
- Could only store *out* edges
  - Figure out *in* edges with increased computation/ time
  - Useful to have both *in* and *out* edges

CONNECTIVITY IN DIRECTED GRAPHS

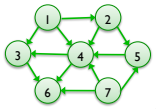
Graph Search

- How does ***reachability*** change with directed graphs?

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

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

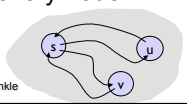
- Rather than paths from *s* to other nodes, find all nodes with paths to *s*

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* → *v* and also a *path* from *v* → *u* (not necessarily a direct edge)
- 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

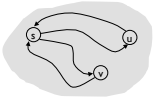
- If *u* and *v* are mutually reachable and *v* and *w* are mutually reachable, then *u* and *w* are mutually reachable.
- Proof. We need to show that there is a path from *u* → *w* and from *w* → *u*.
  - By defn of mutually reachable
    - there is a path *u* → *v* & a path *v* → *u,*
    - a path *v* → *w,* and a path *w* → *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 ⇒
  - 2nd prove ⇐
    - for any nodes *u* and *v,* is there a path *u*→*v* and *v*→*u* ?

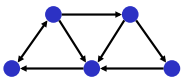
Strong Connectivity

- 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. ⇒ Follows from definition of strongly connected
- Pf. ⇐ 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*

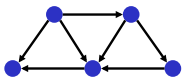


Strong Connectivity Problem

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



strongly connected



not strongly connected

Hint: 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 Grev
  - 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

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

Strong Components

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

Hint: Consider a node in common…

Strong Components

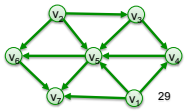
- For any two nodes *s* and *t* in a directed graph, their strong components are either identical or disjoint
- Proof.
  - Consider *v* in both strong components
    - s→ v; v → s; v→t; t→v ➔ t→s, s→t (mutually reachable)
    - As soon as there is one common node, then have identical strong components
  - On the other hand, consider *s* and *t* are not mutually reachable
    - No node *v* that is in the strong component of each
      - What would it mean if there were?

DAGS AND TOPOLOGICAL ORDERING

Directed Acyclic Graphs

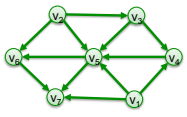
- Def. A ***DAG*** is a directed graph that contains no directed cycles.
- Example. Precedence constraints: edge (v_i, v_j) means v_i must precede v_j
  - Course prerequisite graph: course v_i must be taken before v_j
  - Compilation: module v_i must be compiled before v_j
  - Pipeline of computing jobs: output of job v_i needed to determine input of job v_j

a DAG:



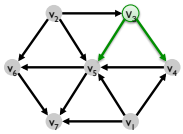
Problem: Valid Ordering

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

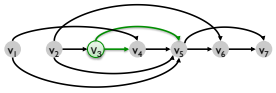


Topological Ordering

- Problem: 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, …, v_n so that for every edge (v_i, v_j) we have i < j.



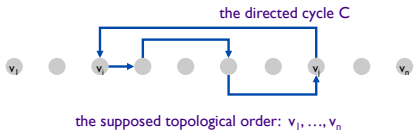
a DAG



a topological ordering
All edges point "forward"

Directed Acyclic Graphs

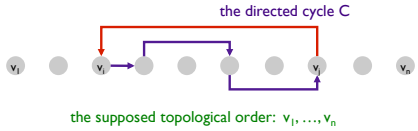
- Lemma. If G has a topological order, then G is a DAG.
- Proof plan: Try to show that G has a cycle



Why isn't this valid?

DAGs & Topological Orderings

- 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
  - 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. ▪

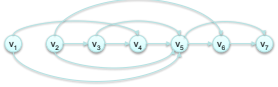


Directed Acyclic Graphs

- Does every DAG have a topological ordering?
  - If so, how do we compute one?

Directed Acyclic Graphs

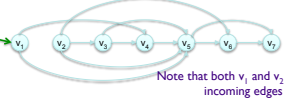
- Does every DAG have a topological ordering?
  - If so, how do we compute one?
- What would we need to be able to create a topological ordering?
  - What are some characteristics of this graph?



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?

Need someplace to start: a node with no incoming edges (no dependencies)



Note that both v_1 and v_2 have no incoming edges

Directed Acyclic Graphs

- Lemma. If G is a DAG, then G has a node with no incoming edges
  - This is our starting point of the topological ordering
- How to prove?

Directed Acyclic Graphs

- Lemma. If G is a DAG, then G has a node with no incoming edges
- Proof idea: consider if there is no node without incoming edges
  - What do we want to show?

To be continued …

PS1 Feedback

- Problem 1: Looking for an induction proof
- Problem 2
  - don't need the Gale-Shapley algorithm to prove, just base on problem defn/description
- Problem 3
  - Algorithm adaptation: need to break/handle ties in G-S
    - Since still using G-S, no strong instabilities
  - Example of weak instability
- Problem 4: Straightforward adaptation of definitions
  - Trying to get you to review the definitions and get more comfortable with them
- Problems 5: Similar to one of the solved exercises

Assignments

- Reading Chapter 3.1-3.5
  - Wikis for Wednesday
- For next Friday: Problem Set 3