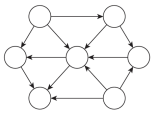
Objectives

- Directed Graphs
- Topological Orderings
- DAGs

Review: Directed Graphs G = (V, E)

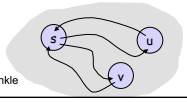
- Edge (u, v) goes from node u to node v



- Representation
  - Maintain both in and out edges of each node

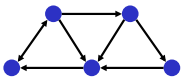
Review: Strong Connectivity

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

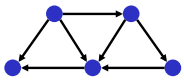


Review: Strong Connectivity Problem

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



strongly connected



not strongly connected

Hint: We can leverage an O(m+n) algorithm

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

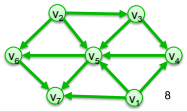
- For any two nodes *s* and *t* in a directed graph, their strong components are either identical or disjoint
- Proof.
  - Consider *v* in both strong components
    - s→ v; v → s; v→t; t→v → t→s, s→t (mutually reachable)
    - As soon as there is one common node, then have identical strong components
  - 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

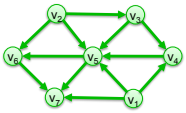
- Def. A ***DAG*** is a directed graph that contains no directed cycles.
- Example. Precedence constraints: edge (v_i, v_j) means v_i must precede v_j
  - Course prerequisite graph: course v_i must be taken before v_j
  - Compilation: module v_i must be compiled before v_j
  - Pipeline of computing jobs: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

a DAG:



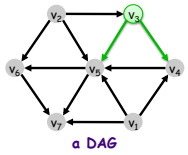
Problem: Valid Ordering

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

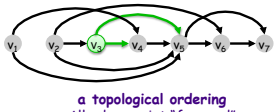


Topological Ordering

- 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), i < j.



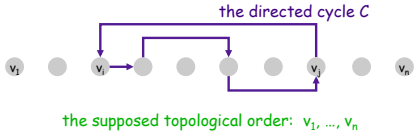
a DAG



a topological ordering
All edges point "forward"

DAGs & Topological Orderings

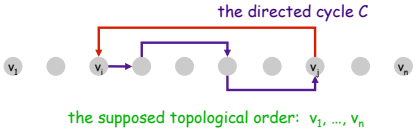
- Lemma. If G has a topological order, then G is a DAG.
- Proof: What if G has a cycle *and* a topological order?



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, …, 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, …, v_n is a topological order, we must have j < i, a contradiction. ▪

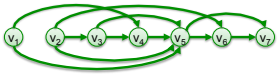


DAGs & Topological Orderings

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

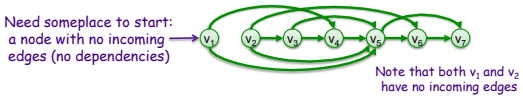
DAGs & Topological Orderings

- 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?



DAGs & Topological Orderings

- 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?



DAGs & Topological Orderings

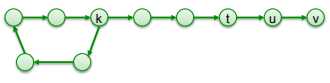
- Lemma. If G is a DAG, then G has a node with no incoming edges
  - This is our starting point of the topological ordering
- How to prove?

DAGs & Topological Orderings

- 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 does that mean?
  - How can we get to a contradiction?

DAGs & Topological Orderings

- Lemma. If G is a DAG, then G has a node with no incoming edges.
- Pf. (by contradiction)
  - Suppose that G is a DAG and every node has at least one incoming edge
  - Pick any node *v*, and follow edges backward from *v*.
    - Since *v* has at least one incoming edge (*u*, *v*), walk backward to *u*
  - Since *u* has at least one incoming edge (*t*, *u*), walk backward to *t*
  - Repeat until we visit a node, say *k*, twice
    - Has to happen at least by *n*+1 steps (Why?)
  - Let C denote the sequence of nodes encountered between successive visits to *k*. C is a cycle. ▪

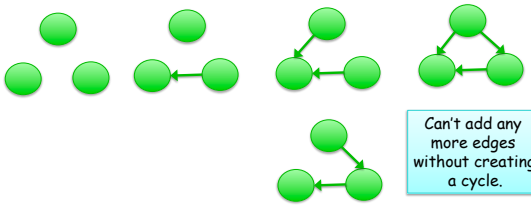


Creating a Topological Order

- Claim: If there is a node with no incoming edges, can create a topological ordering
- Think about a DAG with only one node. What is its topological ordering?
- Only two nodes?
- Three nodes?
  - What are the DAG, TO possibilities?

Topological Order for Three Nodes

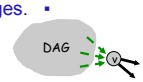
- What are the possibilities?



Can't add any more edges without creating a cycle.

DAGs & Topological Orderings

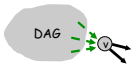
- Lemma. If G is a DAG, then G has a topological ordering.
- Pf. (by induction on n)
  - Base case: true if n = 1
  - Given DAG on n > 1 nodes, find a node v with no incoming edges
  - G - { v } is a DAG, since deleting v cannot create cycles
  - By inductive hypothesis, G - { v } has a topological ordering
  - Place v first in topological ordering; then append nodes of G - { v } in topological order.
  - Valid since v has no incoming edges. ▪



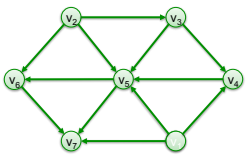
DAGs & Topological Orderings

- Lemma. If G is a DAG, then G has a topological ordering.
- Algorithm:

```
Find a node v with no incoming edges
Order v first
Delete v from G
Recursively compute a topological ordering of G-{v}
      and append this order after v
```

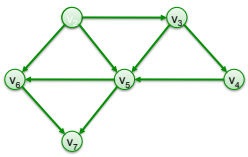


Topological Ordering Algorithm: Example



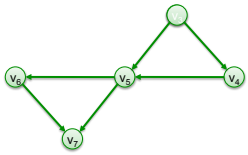
Topological order:

Topological Ordering Algorithm: Example



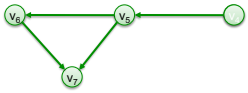
Topological order: v_1

Topological Ordering Algorithm: Example



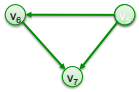
Topological order: v_1, v_2

Topological Ordering Algorithm: Example



Topological order: v_1, v_2, v_3

Topological Ordering Algorithm: Example



Topological order: v_1, v_2, v_3, v_4

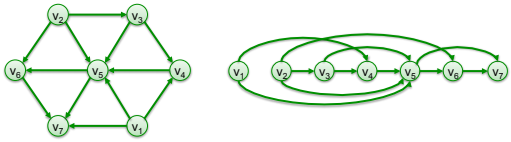
Topological Ordering Algorithm: Example

Topological order: v_1, v_2, v_3, v_4, v_5

Topological Ordering Algorithm: Example

Topological order: v_1, v_2, v_3, v_4, v_5, v_6

Topological Ordering Algorithm: Example



Topological order: v_1, v_2, v_3, v_4, v_5, v_6, v_7.

Topological Order Runtime

- Where are the costs?

```
Find a node v with no incoming edges
Order v first
Delete v from G
Recursively compute a topological ordering of G-{v}
      and append this order after v
```


Topological Order Runtime

- Where are the costs?

```
Find a node v with no incoming edges
Order v first
Delete v from G
Recursively compute a topological ordering of G-{v}
      and append this order after v
```

- Find a node without incoming edges and delete it: **O(n)**
- Repeat on all nodes
- → **$O(n^2)$**

Can we do better?

Topological Sorting Algorithm: Running Time

- Theorem. Find a topological order in O(m + n) time
- Pf.
  - Maintain the following information:
    - count[w] = remaining number of incoming edges
    - S = set of remaining nodes with no incoming edges
  - Initialization: O(m + n) via single scan through graph
  - Update: to delete v
    - Remove v from S
    - Decrement count[w] for all edges from v to w
      - Add w to S if count[w] hits 0
    - O(1) per edge ▪

Extra Credit Opportunity

- **Talk: "A Mathematician's Year on Capitol Hill"**
- Katherine Crowley, W&L Mathematics Department
- Time: Wednesday, February 3, 3:30 p.m.
- Place: Robinson Hall, Room 6
- 10 points towards problem set grade

Assignments

- Finish reading Chapter 3
  - Wikis for Wednesday
- For Friday: Problem Set 3
- Friday: Handout exam
  - Next Wednesday – work period
    - Ask questions