Objectives

- Directed Graphs
- Topological Orderings of DAGs

Graph Summary So Far

- What do we know about graphs?

Graph Summary So Far

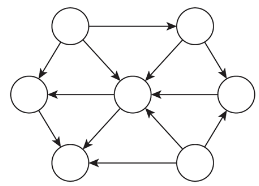
- What do we know about graphs?
  - Representation: Adjacency List, Space O(n+m)
  - Connectivity
    - BFS, DFS – O(n+m)
- Can apply BFS for Bipartite

Second verse, similar to the first.
But directed!

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

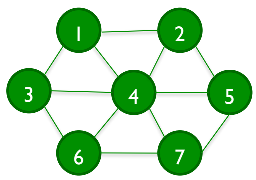
- For each node, keep track of
  - Out edges (where links go)
  - In edges (from where links come in)
  - Space required?
- Could only store *out* edges
  - Figure out *in* edges with increased computation/time
  - Useful to have both *in* and *out* edges

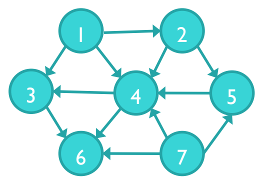
Rock Paper Scissors Lizard Spock

CONNECTIVITY IN DIRECTED GRAPHS

Graph Search

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

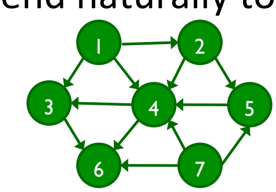




- Example: Web crawler
  1. Start from web page s.
  2. Find all web pages linked from s, either directly or indirectly.

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

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

Problem/Solution

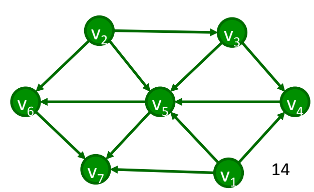
- Problem. Find all nodes with paths **to** *s*
- Solution. Run BFS on *in edges* instead of out edges

DAGS AND TOPOLOGICAL ORDERING

Directed Acyclic Graphs

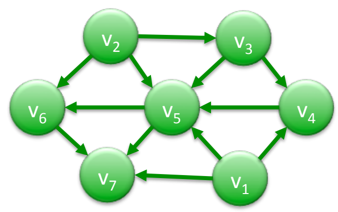
- Def. A ***DAG*** is a directed graph that contains no directed cycles.
- Example. Precedence constraints: edge (v_i, v_j) means v_i must precede v_j
  - Course prerequisite graph: course v_i must be taken before v_j
  - Compilation: module v_i must be compiled before v_j
  - Pipeline of computing jobs: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

a DAG:



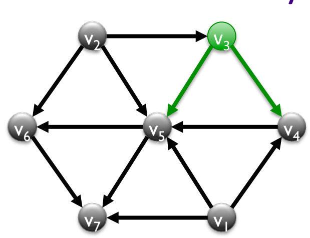
Problem: Valid Ordering

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

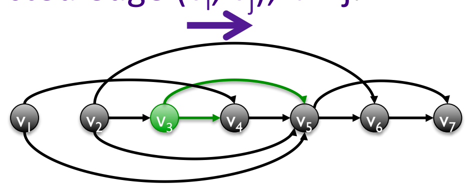


Topological Ordering

- 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 $v_1, v_2, \ldots, v_n$ such that for every directed edge (v_i, v_j), $i < j$.



a DAG



a topological ordering
All edges point "forward"

Coordinating labeling of nodes, but numbering is not known for just DAG

Topological Ordering Example

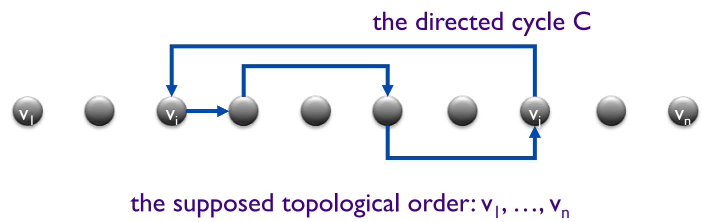
- Given a set of tasks with dependencies, what is a valid order in which the tasks could be performed?
  - Example: Course prerequisites
    - Values of the nodes vs. their ids
- A ***topological order*** of a directed graph $G = (V, E)$ is an ordering of its nodes as $v_1, v_2, \ldots, v_n$ such that for every directed edge (v_i, v_j), $i < j$.

Towards a Solution

- Start by showing that if G has a topological order, then G is a DAG
- Eventually, we'll show the other direction: if G is a DAG, then G has a topological order

Directed Acyclic Graphs

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

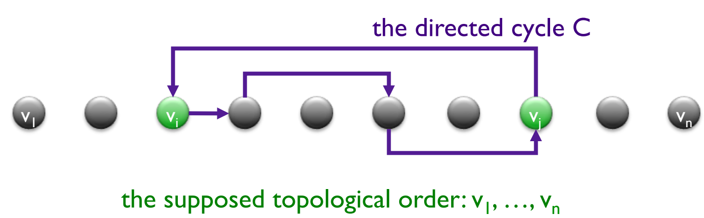


Why isn't this valid?

DAGs & Topological Orderings

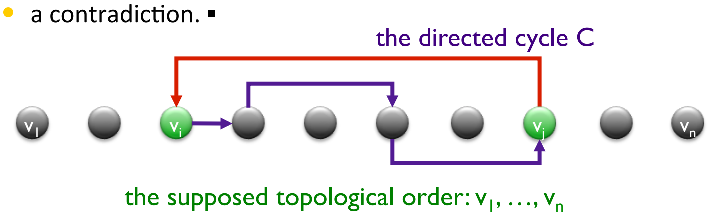
- Lemma. If G has a topological order, then G is a DAG.
- Pf. (by contradiction)
  - Suppose that G has a topological order $v_1, \ldots, v_n$ and that G also has a directed cycle C.

What can we say about that cycle and the nodes, edges in the cycle?



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. ▪

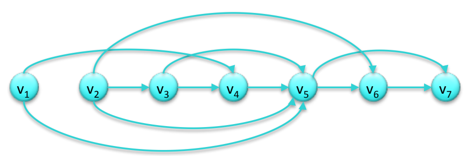


Directed Acyclic Graphs

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

Directed Acyclic Graphs

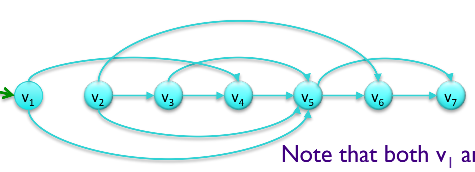
- Does every DAG have a topological ordering?
  - If so, how do we compute one?
- What do we need to be able to create a topological ordering?
  - What are some characteristics of this graph?



Directed Acyclic Graphs

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

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



Note that both v_1 and v_2
have no incoming edges

Towards a Topological Ordering

Goal: Find an algorithm for finding the TO
Idea: 1st node is one with no incoming edges

Do we know there is always a node with no incoming edges?

Towards a Topological Ordering

- 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?

Towards a Topological Ordering

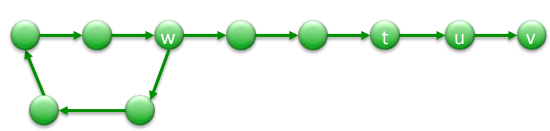
- Lemma. If G is a DAG,
  then G has a node with no incoming edges

- Proof idea: Consider if there is no node without incoming edges
  - *Restated: All nodes have incoming edges.*
  - What contradiction are we looking for?

Feb 5, 2018 CSCI211 - Sprenkle 27

Towards a Topological Ordering

- 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), we can walk backward to u
  - Since u has at least one incoming edge (t, u), we can walk backward to t
  - Repeat until we visit a node, say *w*, twice
    - Has to happen at least by step n+1 (Why?)
  - Let C denote the sequence of nodes encountered between successive visits to *w*. C is a cycle, which is a contradiction to G is a DAG ▪



Feb 5, 2018 CSCI211 - Sprenkle 28

Putting it all together: Creating a topological order

Ideas?

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
```

How do we know this works?

Directed Acyclic Graphs

- Lemma. If G is a DAG, then G has a topological ordering.
- Pf. (by induction on n)
  - Base case:

Directed Acyclic Graphs

- 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

Directed Acyclic Graphs

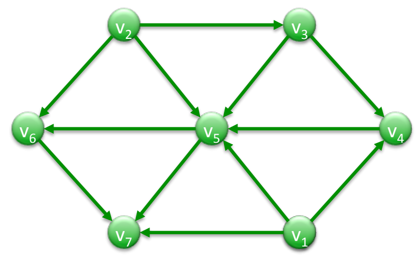
- Lemma. If G is a DAG, then G has a topological ordering.
- Pf. (by induction on n)
  - Base case: true if n = 1
  - Given DAG on n > 1 nodes, find a node v with no incoming edges
  - G - { v } is a DAG because deleting v cannot create cycles
  - Also know, by inductive hypothesis, G - { v } has a topological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

Topological Ordering Algorithm

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

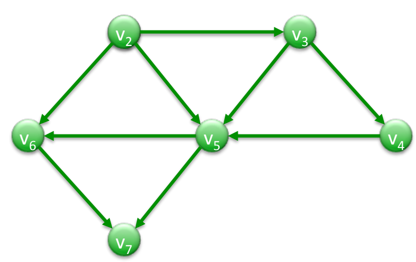
```
Find a node v with no incoming edges
Order v first
Delete v from G
Recursively compute a topological ordering of G-{v}
      and append this order after v
```

Topological Ordering Algorithm: Example



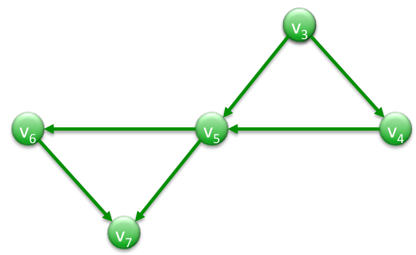
Topological order:

Topological Ordering Algorithm: Example



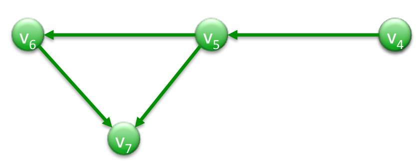
Topological order: v_1

Topological Ordering Algorithm: Example



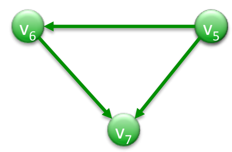
Topological order: v_1, v_2

Topological Ordering Algorithm: Example



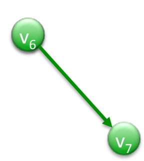
Topological order: v_1, v_2, v_3

Topological Ordering Algorithm: Example



Topological order: v_1, v_2, v_3, v_4

Topological Ordering Algorithm: Example

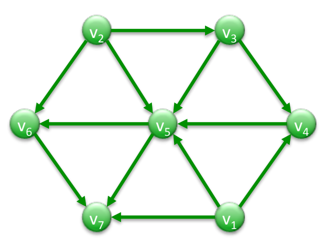


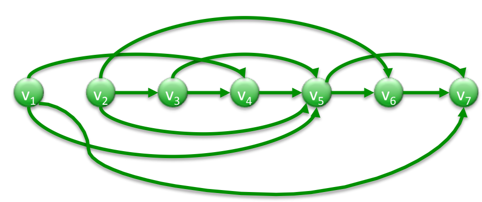
Topological order: v_1, v_2, v_3, v_4, v_5

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

```
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
```

- Where are the costs?
- How would we implement?

Topological Order Runtime

```
Find a node v with no incoming edges  O(n)
Order v first O(1)
Delete v from G  O(n)
Recursively compute a topological ordering of G-{v}  O(n)
      and append this order after v O(1)
```

- 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
  - Algorithm:
    - Select a node v from S, remove v from S
    - Decrement count[w] for all edges from v to w
      - Add w to S if count[w] = 0

Feb 5, 2018 CSCI211 - Sprenkle 45

Looking Ahead

- Wiki due Tuesday at 11:59 p.m.
  - Sections 3.2-3.6
- Problem Set 4 due Friday

Feb 5, 2018 CSCI211 - Sprenkle 46