CS 2100

Graphs:
topological sort

Recap

- HW8 is graded
- HW4 in git (please check!)

→ check blackboard

- Practice finals (will bring more Wed.)
- No lab this week (office hours/work time)

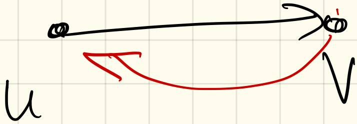
Today: Directed graph searching

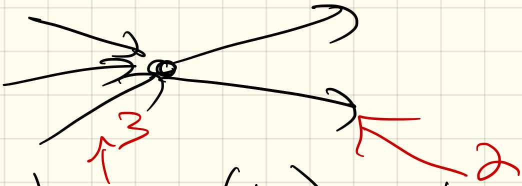
Recall:

$\neq (v,u)$

$(u,v) \in E$:

not $\{u,v\}$





$\text{indegree}(u) \neq \text{outdegree}(u)$

Degree-Sum Formula (revised):

$$\sum_v \text{indeg}(v) + \sum_v \text{outdeg}(v) = 2|E|$$

First: traversal

We can modify BFS/DFS code easily:

stack or queue

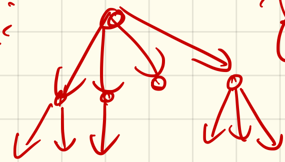
```
TRAVERSE(s):
    put s into the bag
    while the bag is not empty
        take v from the bag
        if v is unmarked
            mark v
            for each edge vw
                put w into the bag
```

O(1) for each incoming edge $\Rightarrow O(\text{indeg}(v))$

outgoing edge

$O(\text{outdeg}(v))$ runtime

BFS:



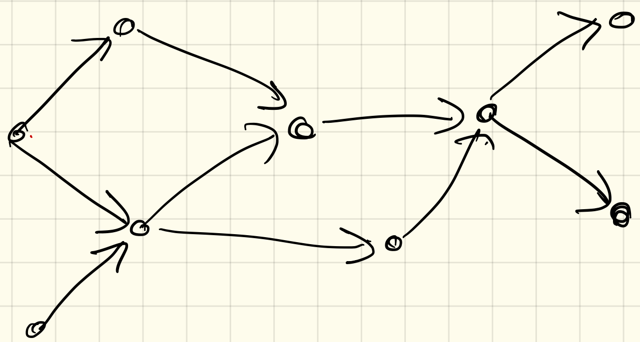
Runtime: $O\left(n + \sum_v \text{indeg}(v) + \sum_v \text{outdeg}(v)\right)$

$= O(m+n)$

Special interesting case:

Directed acyclic graphs: DAG

graphs with no directed cycles



These are specialized, but still useful:

Ex: prereqs/classes in a major

CS1300 → CS2100 → ...

Discrete Math →

Ex (cont):

Inheritance or makefiles in C++

BinaryTree.h → BST.h → AVL.h

BinaryTree.h → decode.cpp

BitStreams.h → decode.cpp

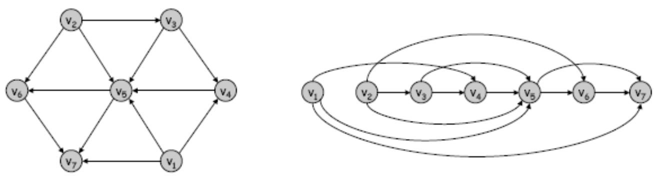
Ex: In software engineering, dependency charts

designing modular systems

Many more...

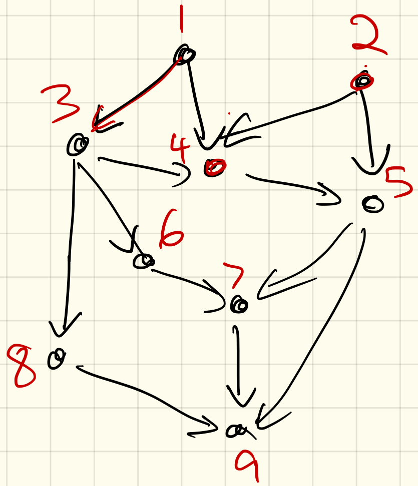
Definition: Given G = a directed graph with n vertices, a *topological ordering* of G is a vertex list: $v_1, v_2, \dots, v_n$ such that for every edge $(v_i, v_j) \in G$, $i < j$.

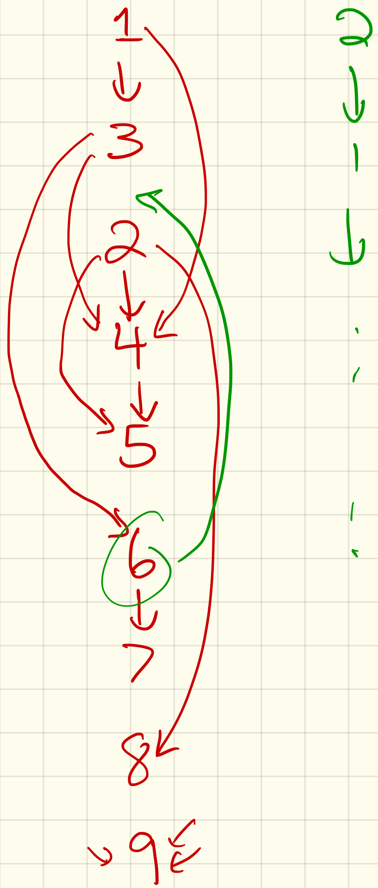
[In other words, edges only point forward.]

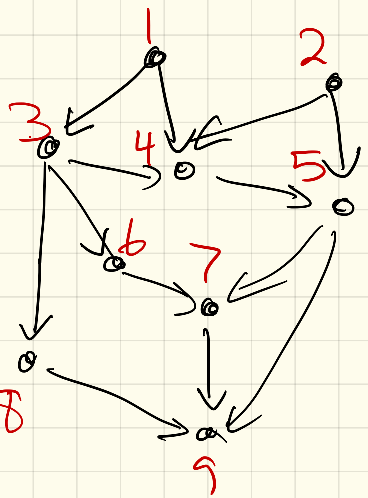


a DAG　　　a topological ordering

Note: these are <u>not</u> unique.







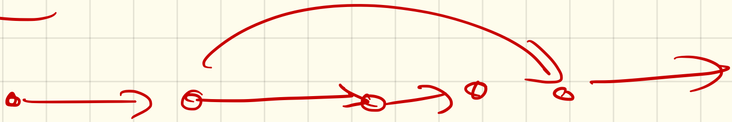
__Proposition:__ G has a topological ordering

$$\Longleftrightarrow$$

G is acyclic.

__"Proof":__

$\Rightarrow$



Sppse top ordering. Then no "backwards" edges, so can't have a directed cycle.

$\Leftarrow$: Start w/ source put it first + delete. repeat.

Leads to algorithm:

Sort by indegree.

Someone must have indegree 0.

Put him next, delete & update indegrees.

(Repeat)

Or – nicer:

this is not O(1) time

```
TopologicalSort(G):
  n ← |V|
  for i ← 1 to n
      v ← any source in G
      S[i] ← v
      delete v and all edges leaving v
  return S[1..n]
```

```
TopologicalSort(G):
  n ← |V|
  for i ← n down to 1
      v ← any sink in G
      S[1] ← v
      delete v and all edges entering v
  return S[1..n]
```

Runtime: $O(n \log n) + O(n)$

$= O(n \log n)$

End of graphs:

But way more to do with them!

- Routing
- Drawing & layouts
- Computing subgraphs

⋮

(Go take 3100 & you'll see more)

Next time:

Brief overview of sets

Go take course eval!