Quick notes

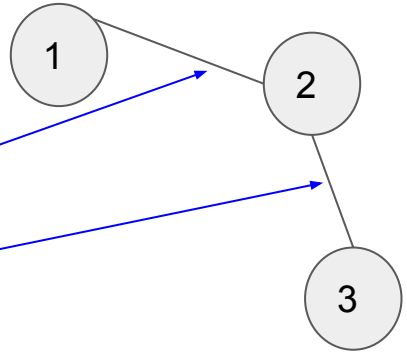
- Check Piazza
- Reminders:
  - Assign8 due 4th Dec
  - Assign9 due 11th Dec (NO LATE DAYS!!)
  - Weekly quiz
- Assignment 10
  - **Email me your topic**
  - Create presentation (<10 min, semi-strict) + brief summary (<200 words)
    - Must present to class (email me if you're in a different timezone)
    - You can pre-record (e.g. youtube)
  - Order is here (tentative), let me know ASAP if you are unable to present.
    - May have to shift up/down depending on time
  - https://docs.google.com/spreadsheets/d/1rcPnnaa2zH8_3fi3qSLgCKRDdwKgqML32HocBnEv6q0/

Feedback / Office Hours / Other

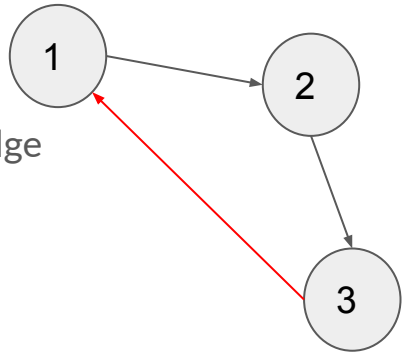
- Tameez Latib
  - tameezlatib@gmail.com, please add “CS35L” to the subject line
  - Office Hours: Monday **5pm-7pm Nov 30** (or by appointment)
  - Feedback: https://forms.gle/6kcJ2aJtzAzFMhHQ7 (anonymous google form)
- If you guys are stressed out:
  - CAPS (https://www.counseling.ucla.edu/)
    - Free with UC ship
- Week 9!
  - We’re done with coursework after today,
  - We will have final review lectures starting Wednesday
- About the final:
  - 24 hour period to take the final
  - Once you download/open the file, your 3 hours starts (**DO NOT DOWNLOAD FOR LATER**)
    - You MUST turn it in around 3 hours after you download it.

What is a graph (G)

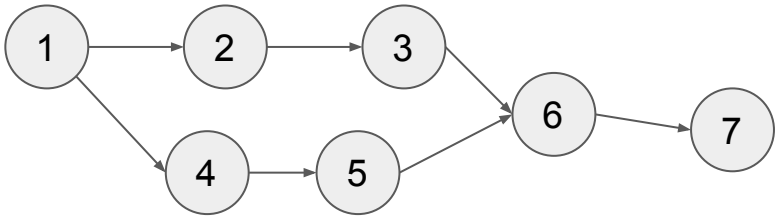
- G = (Vertices, Edges) = (V, E)
- Vertices are nodes
- Edges connect nodes
- E.g.
  - V = [1, 2, 3]
  - E =[(1,2), (2,3)]



- Graph properties
  - Directed (D): edges have direction
  - Acyclic (A): No cycles. Graph on right is cyclic
    - If we want it to be acyclic, take out red edge



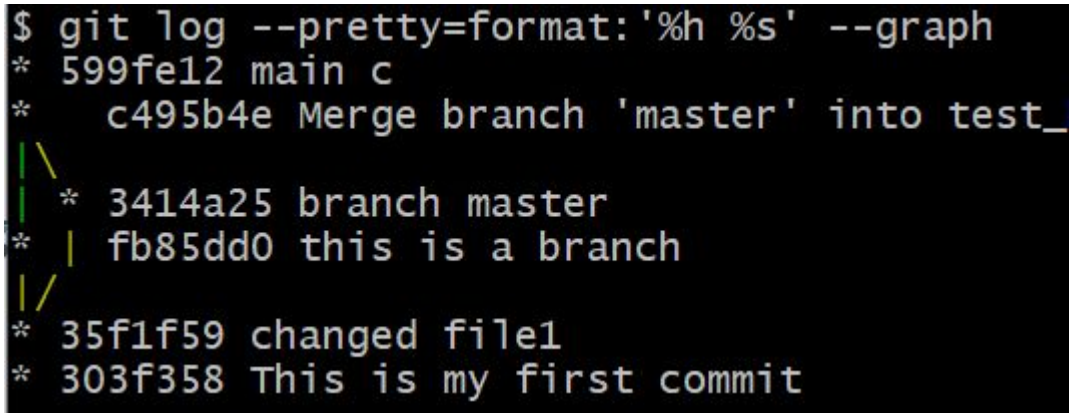
Git uses DAG (Directed Acyclic Graph)



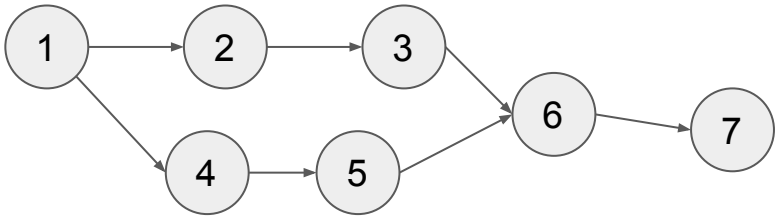
- Why DAG?
  - Can't go back in time, commits always go forward
- Question:
  - Given ONLY V and E, can we construct the above graph? How?
  - V = [1, 2, ... 7]
  - E = [(1->2), (2->3), (3->6), (6->7), (1->4), (4->5), (5->6)]

Topological sort

- Note git only knows V and E
  - But if we use the below git log command, we are able to create a nice graph
- This is “topological sort”
  - There are multiple valid sorts! (We can swap 3414a25 and fb85dd0 if we want)
- (For math majors: DAG is partially order set)
  - We cannot compare 3414a25 and fb85dd0, we get to choose which comes first

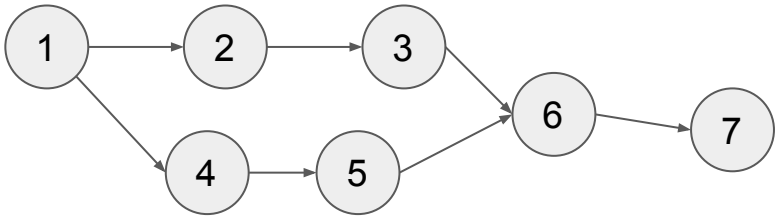


Topological sort (description)



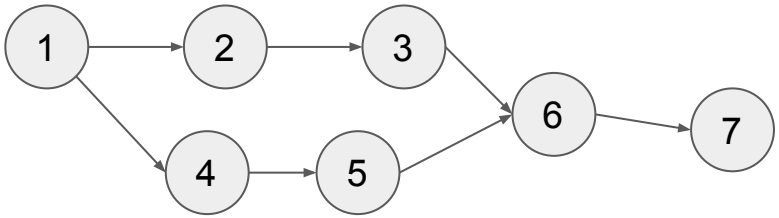
- Generally, we **have** some DAG (V, E)
- We **want** a linear output, v_1, v_2 ,.... v_n
  - And there is no path from v_{i+j} to v_i
    - Equivalent to “There is path from v_i to v_{i+j} or they are on separate branches”
- From the above DAG, some possibilities are
  - “1 2 3 4 5 6 7”
  - “1 4 2 3 5 6 7”
- Note that “1 5 2 3 4 6 7” is NOT okay
  - We have a path from 4->5, but 5 comes before 4 in this list

Topological sort (how/intuition)



- By looking at DAG we know a few things:
  - First element (v_1) MUST be 1 (it is the root note)
    - $v_1 = 1$
  - v_1 connects to 2 and 4, so it makes sense for v_2 to be either 2 or 4
    - $v_2 = 2$
  - We know that v_3 should be 3 or 4 (we have all ancestors of both 3 and 4)
    - $v_3 = 3$
  - Note that v_4 can't be 6, because 6 is connected to 5, and we haven't added 5 to our list yet
  - So now we take the other path (4, 5)
    - $V_4 = 4$

Topological sort (algorithm)

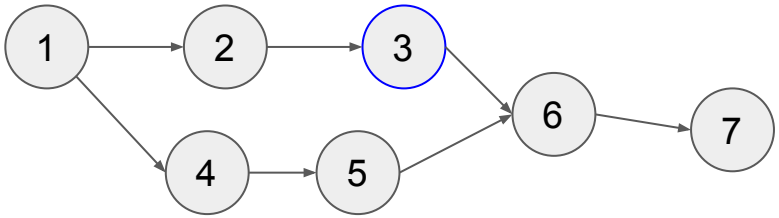


- Example, let n=3
- Blue circle = temporary mark
- Red circle = permanent mark
- L = []
- visit(3)
  - Temp mark n=3
  - The only n->m is 3->6, so we will visit(6)
- https://en.wikipedia.org/wiki/Topological_sorting

DFS algorithm. Last slide was more similar to Kahn

```
L ← Empty list that will contain the sorted nodes
while exists nodes without a permanent mark do
    select an unmarked node n
    visit(n)

function visit(node n)
    if n has a permanent mark then
        return
    if n has a temporary mark then
        stop   (not a DAG)

    mark n with a temporary mark

    for each node m with an edge from n to m do
        visit(m)

    remove temporary mark from n
    mark n with a permanent mark
    add n to head of L
```

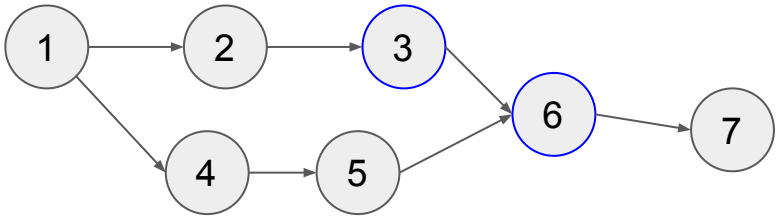
Topological sort (algorithm)



- DAG before visit(6)
- L before visit(6): []
- visit(6)
  - Temp mark n=6
  - Only have 6->7 so will visit(7)

```
L ← Empty list that will contain the sorted nodes
while exists nodes without a permanent mark do
    select an unmarked node n
    visit(n)

function visit(node n)
    if n has a permanent mark then
        return
    if n has a temporary mark then
        stop   (not a DAG)

    mark n with a temporary mark

    for each node m with an edge from n to m do
        visit(m)

    remove temporary mark from n
    mark n with a permanent mark
    add n to head of L
```

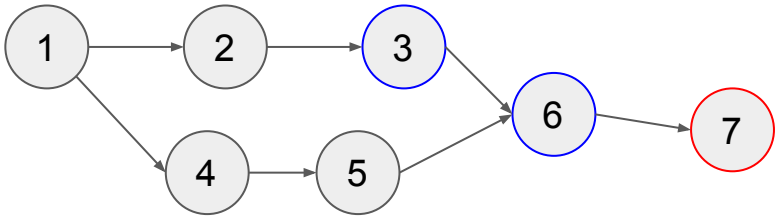
Topological sort (algorithm)



- DAG before visit(7)
- L before visit(7): []
- visit(7)
  - Temp mark n=7
  - No 7->_
  - Permanent mark n=7
  - 7 added to L

```
L ← Empty list that will contain the sorted nodes
while exists nodes without a permanent mark do
    select an unmarked node n
    visit(n)

function visit(node n)
    if n has a permanent mark then
        return
    if n has a temporary mark then
        stop   (not a DAG)

    mark n with a temporary mark

    for each node m with an edge from n to m do
        visit(m)

    remove temporary mark from n
    mark n with a permanent mark
    add n to head of L
```

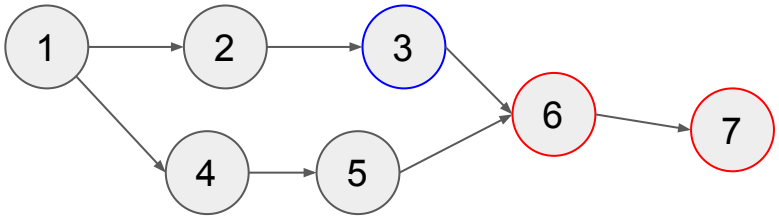
Topological sort (algorithm)



- DAG after visit(7)
- L after visit(7): [7]
- Back to visit(6),
  - Permanent mark n=6
  - 6 added to L

```
L ← Empty list that will contain the sorted nodes
while exists nodes without a permanent mark do
    select an unmarked node n
    visit(n)

function visit(node n)
    if n has a permanent mark then
        return
    if n has a temporary mark then
        stop   (not a DAG)

    mark n with a temporary mark

    for each node m with an edge from n to m do
        visit(m)

    remove temporary mark from n
    mark n with a permanent mark
    add n to head of L
```

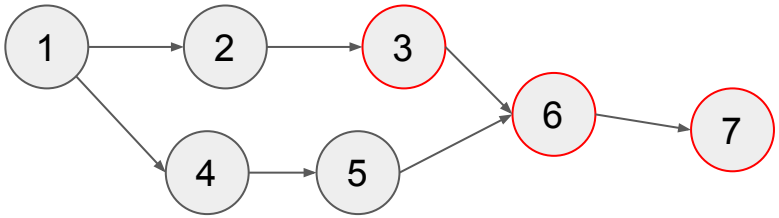
Topological sort (algorithm)



- DAG after visit(6)
- L after visit(6): [6, 7]
- Back to visit(3),
  - Permanent mark n=3
  - 3 added to L

```
L ← Empty list that will contain the sorted nodes
while exists nodes without a permanent mark do
    select an unmarked node n
    visit(n)

function visit(node n)
    if n has a permanent mark then
        return
    if n has a temporary mark then
        stop   (not a DAG)

    mark n with a temporary mark

    for each node m with an edge from n to m do
        visit(m)

    remove temporary mark from n
    mark n with a permanent mark
    add n to head of L
```

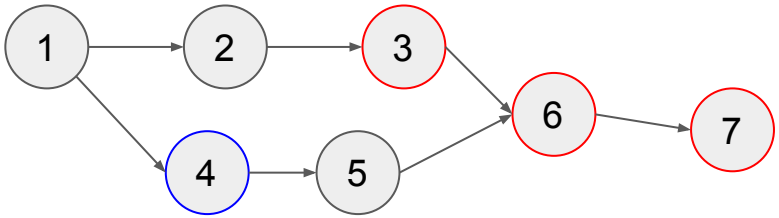
Topological sort (algorithm)



- DAG after visit(3)
- L after visit(3): [3, 6, 7]
- Now we choose a new node, e.g. 4
- Visit(4)
  - Temp mark 4
  - visit(5)

```
L ← Empty list that will contain the sorted nodes
while exists nodes without a permanent mark do
    select an unmarked node n
    visit(n)

function visit(node n)
    if n has a permanent mark then
        return
    if n has a temporary mark then
        stop   (not a DAG)

    mark n with a temporary mark

    for each node m with an edge from n to m do
        visit(m)

    remove temporary mark from n
    mark n with a permanent mark
    add n to head of L
```

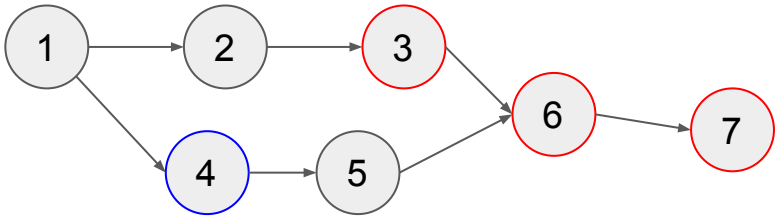
Topological sort (algorithm)



- DAG before visit(5)
- L before visit(5): [3, 6, 7]
- Visit(5)
  - Temp mark 5
  - visit(6)
    - 6 has perm mark, so return
  - Perm mark 5
  - Add 5 to L

```
L ← Empty list that will contain the sorted nodes
while exists nodes without a permanent mark do
    select an unmarked node n
    visit(n)

function visit(node n)
    if n has a permanent mark then
        return
    if n has a temporary mark then
        stop   (not a DAG)

    mark n with a temporary mark

    for each node m with an edge from n to m do
        visit(m)

    remove temporary mark from n
    mark n with a permanent mark
    add n to head of L
```

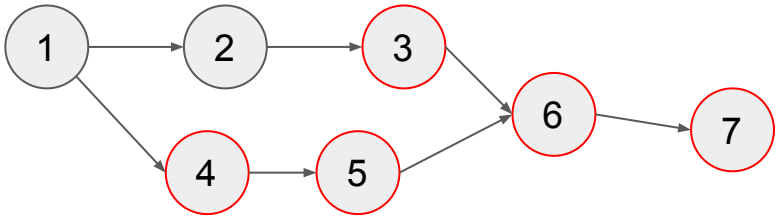
Topological sort (algorithm)



- DAG after visit(5)
- L after visit(5): [5, 3, 6, 7]
- Back to visit(4)
  - Perm mark 4
  - Add 4 to L

```
L ← Empty list that will contain the sorted nodes
while exists nodes without a permanent mark do
    select an unmarked node n
    visit(n)

function visit(node n)
    if n has a permanent mark then
        return
    if n has a temporary mark then
        stop   (not a DAG)

    mark n with a temporary mark

    for each node m with an edge from n to m do
        visit(m)

    remove temporary mark from n
    mark n with a permanent mark
    add n to head of L
```

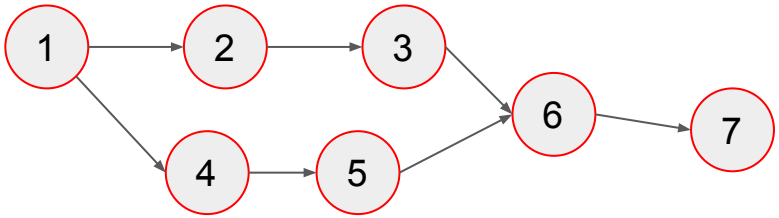
Topological sort (algorithm)



- DAG after visit(4)
- L after visit(4): [4, 5, 3, 6, 7]
- New node, visit(1)
  - Temp mark 1
  - visit(2)
    - Temp mark 2
      - visit(3) -> return
    - Perm mark + add 2 to L
  - Perm mark + add 1 to L

```
L ← Empty list that will contain the sorted nodes
while exists nodes without a permanent mark do
    select an unmarked node n
    visit(n)

function visit(node n)
    if n has a permanent mark then
        return
    if n has a temporary mark then
        stop   (not a DAG)

    mark n with a temporary mark

    for each node m with an edge from n to m do
        visit(m)

    remove temporary mark from n
    mark n with a permanent mark
    add n to head of L
```

Topological sort (algorithm)



- Final DAG
  - All nodes marked, so we end algorithm
- Final L [1, 2, 4, 5, 3, 6, 7]
  - Note this is a valid topological sort

```
L ← Empty list that will contain the sorted nodes
while exists nodes without a permanent mark do
    select an unmarked node n
    visit(n)

function visit(node n)
    if n has a permanent mark then
        return
    if n has a temporary mark then
        stop   (not a DAG)

    mark n with a temporary mark

    for each node m with an edge from n to m do
        visit(m)

    remove temporary mark from n
    mark n with a permanent mark
    add n to head of L
```

Questions??