

Depth-First Search

Note Title

12/4/2007

- Review breadth-first search (bfs)
- Present depth-first search (dfs)
- Present topological sort algorithm

Next Time: Garbage Collection

Tuesday: Review

BFs(V source)

For each $u \in V$
 $u.\text{discovered} = \text{false}$
 $u.\text{dist} = \infty$
 $u.\text{parentEdge} = \text{null}$
 $\text{source}.\text{discovered} = \text{true}$
 $\text{source}.\text{dist} = 0$

initialization

$\text{queue}.\text{enqueue}(\text{source})$

while (!queue.isEmpty())

$u = \text{q}.\text{dequeue}()$

for each edge e in outgoing edges from u

$v = e.\text{dest}$

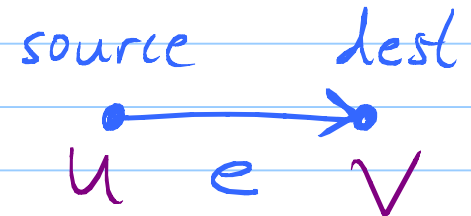
if (! $v.\text{discovered}$)

$v.\text{discovered} = \text{true}$

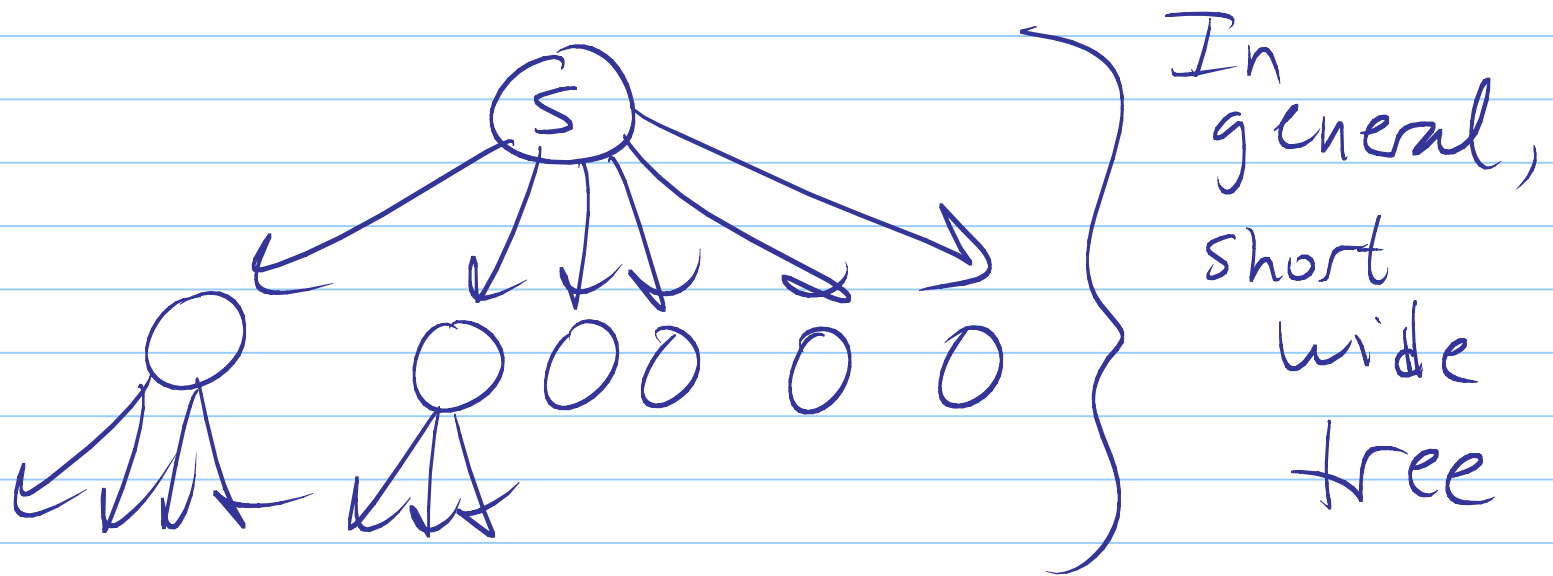
$v.\text{dist} = u.\text{dist} + 1$

$v.\text{parentEdge} = e$

$\text{queue}.\text{enqueue}(v)$



Called breadth-first search
because of how search
proceeds



depth-first search

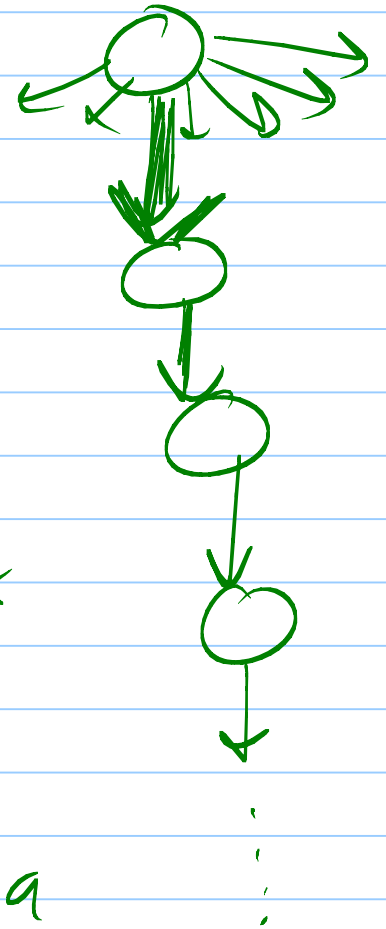
Use recursion

1. Use stack vs queue

2. Continue (re-starting if needed) until all vertices are visited

3. Keep "time" when each vertex discovered + finish processing all outgoing edges

4. Replace boolean discovered by a 3-valued variable, color



false
for
discovered

color white - not yet discovered
discovery time (not on stack yet)

color gray - on stack but not
done processing
finishing time

color black - done processing
(finished)

top-level dfs (in a graph class)

dfs()

color all vertices white + set parent to null
time = 1

for each vertex u in graph

if ($u.color == white$)

dfsVisit(u)

u is
root

} each call
here will
create on
tree in
dfs forest

dfsVisit (Vertex u)

u. color = gray

u. discoveryTime = time++ $u \rightarrow v$

for all vertices v reachable from u
by an edge

if (v.color == white)

v.parent = u

dfsVisit (v)

if (v.color == gray)

what does this tell you? you found
a cycle

u. color = black

u. finishingTime = time++

(can be constructed
using parent
edges)

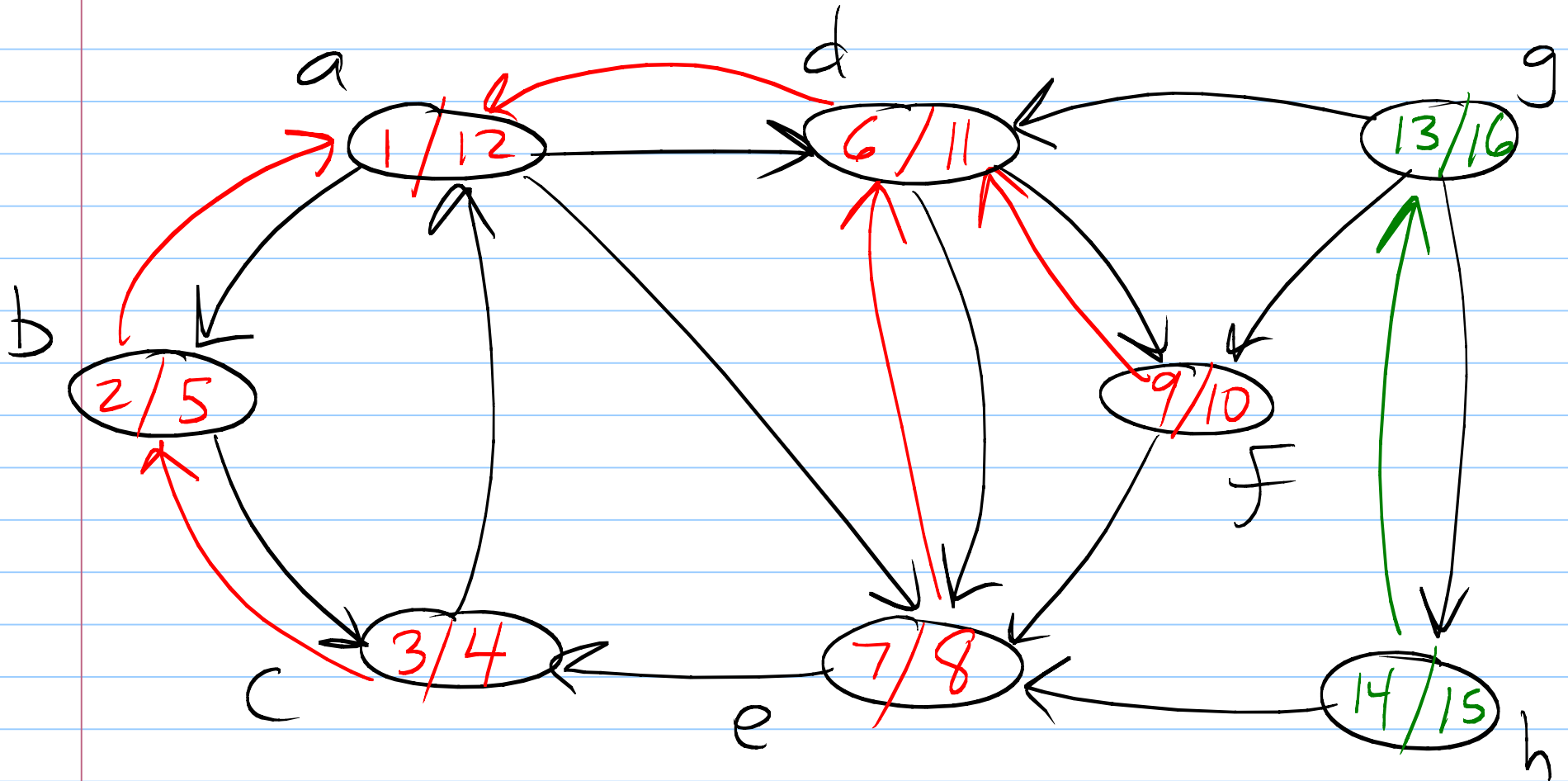
○
white

○
|
grey

○
| #
black

→
parent

Stack
~~⌘~~ ~~⌘~~ ~~⌘~~ ~~⌘~~ ~~⌘~~
bottom top



Time complexity

$O(m+n)$ } with adj
list

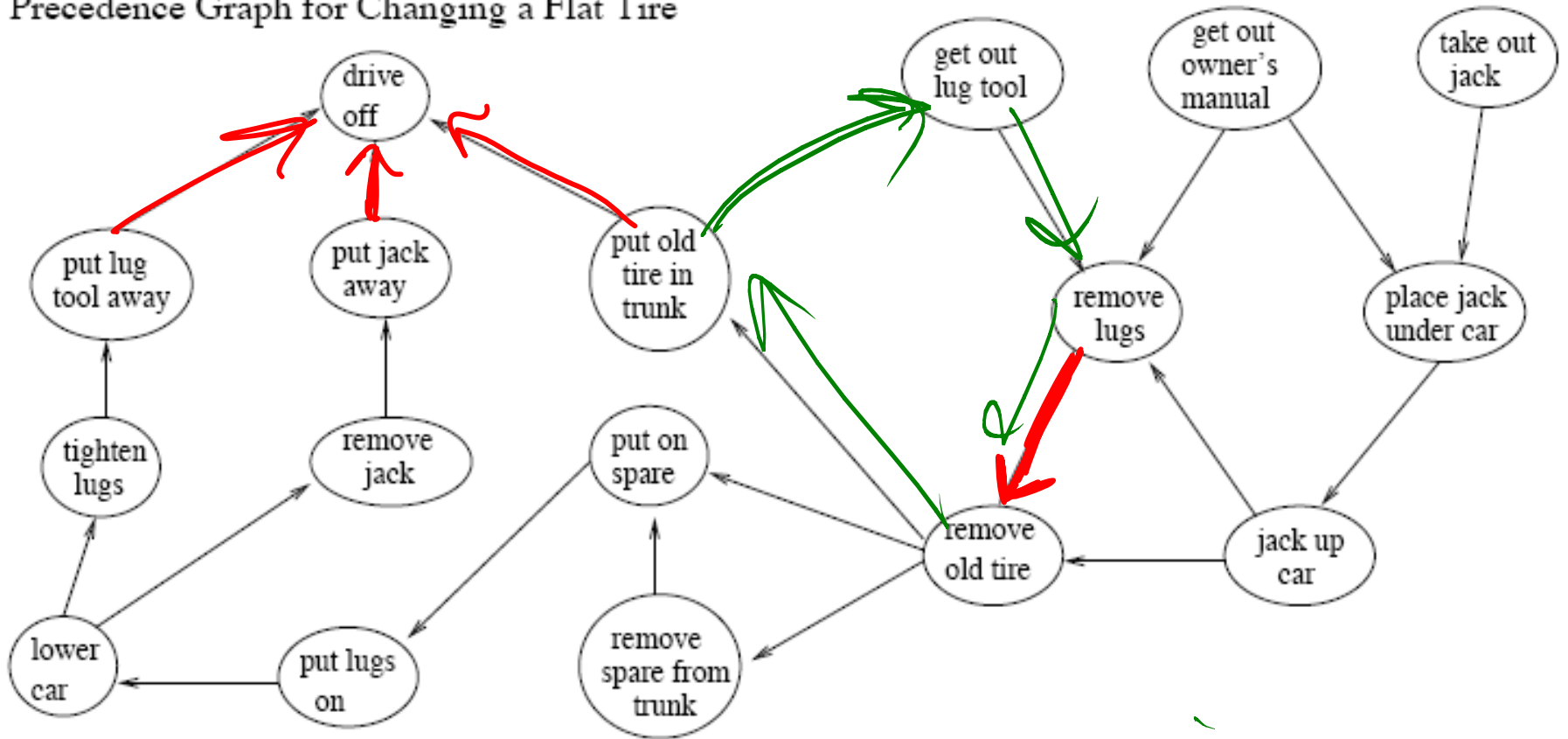
call dfs visit exactly once for
each vertex.

dominant cost is iterating over
outgoing edges

Task Scheduling

$a \rightarrow b$
a must precede b

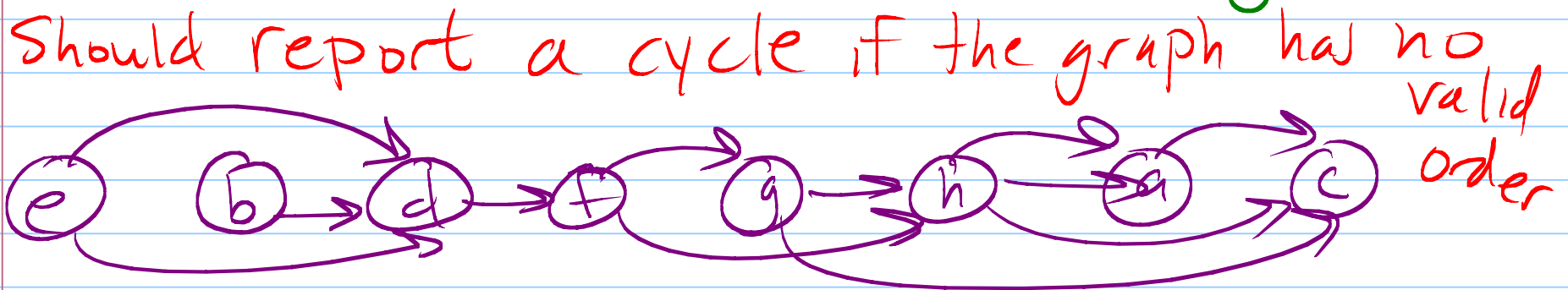
Precedence Graph for Changing a Flat Tire

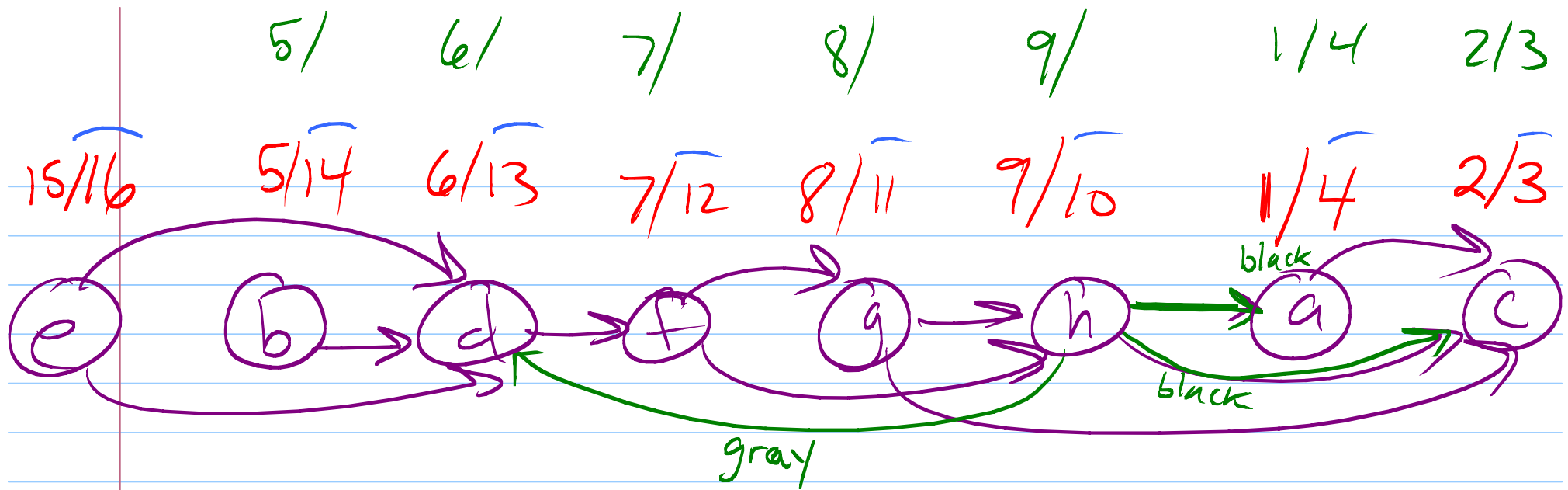


Topological Sort

Find an ordering of vertices in a directed acyclic graph so that if edge $u \rightarrow v$ then u precedes v in ordering

no directed cycles





Observation: valid topological order is defined by the reverse order of finishing times

dfs visit call (w)

IF edge $u \rightarrow v$ we are looking at & v is gray, there is a cycle.

Stack top u

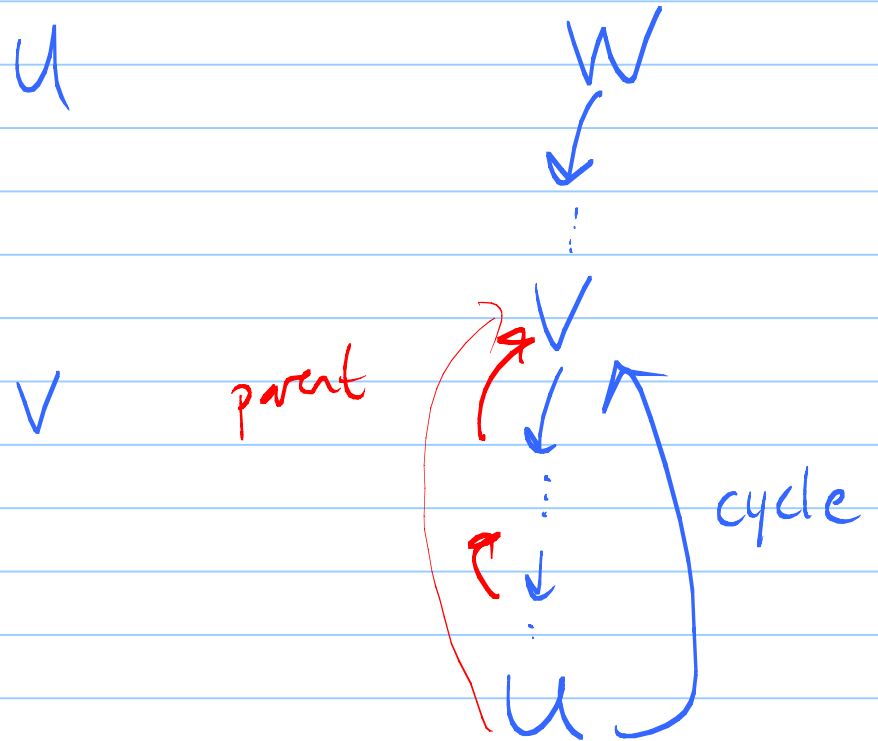
on
stack

when
discovered
put on stack

when finished

taken off
stack

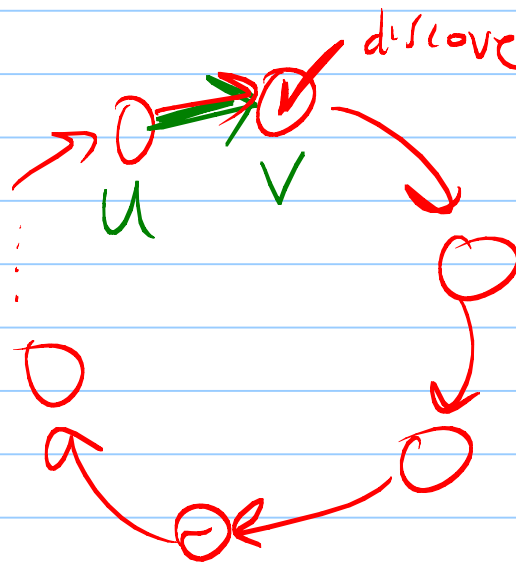
bottom



If there is a directed cycle,
then at some point an edge
will be encountered that goes
gray vertex

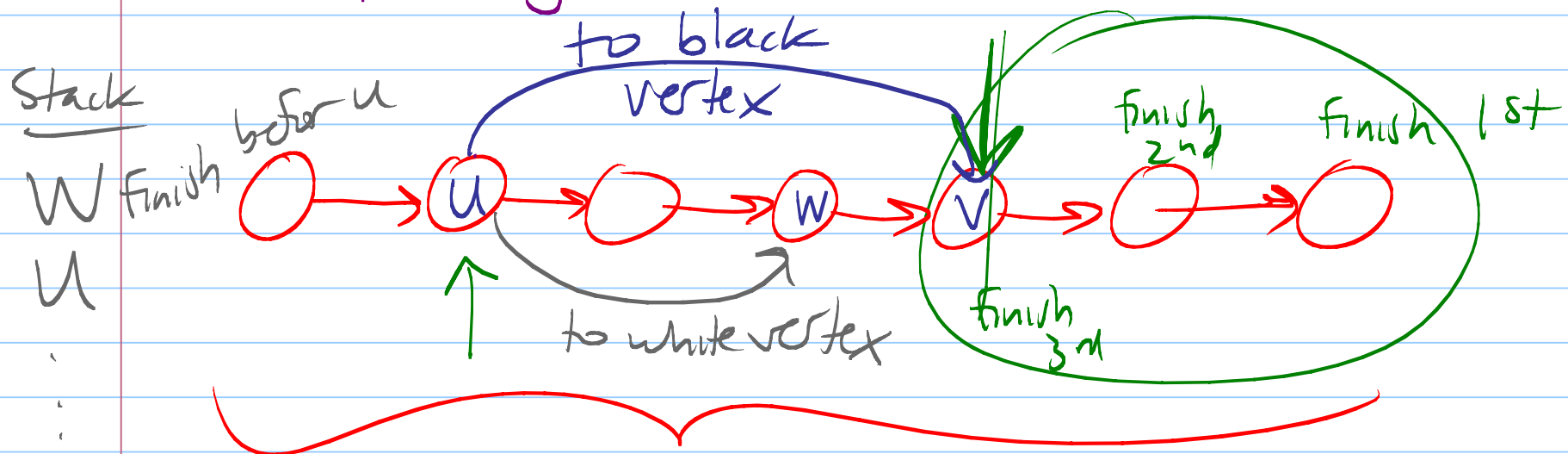
$u \rightarrow v$
grey

cycle which
first has
discovered
vertex



stack
u
v grey

IF graph is acyclic, reverse order of finishing times is a valid topological order



suppose I've drawn in a valid order

Strongly Connected Components.

sets of vertices as big as possible
where each vertex in the set is
reachable from all others

