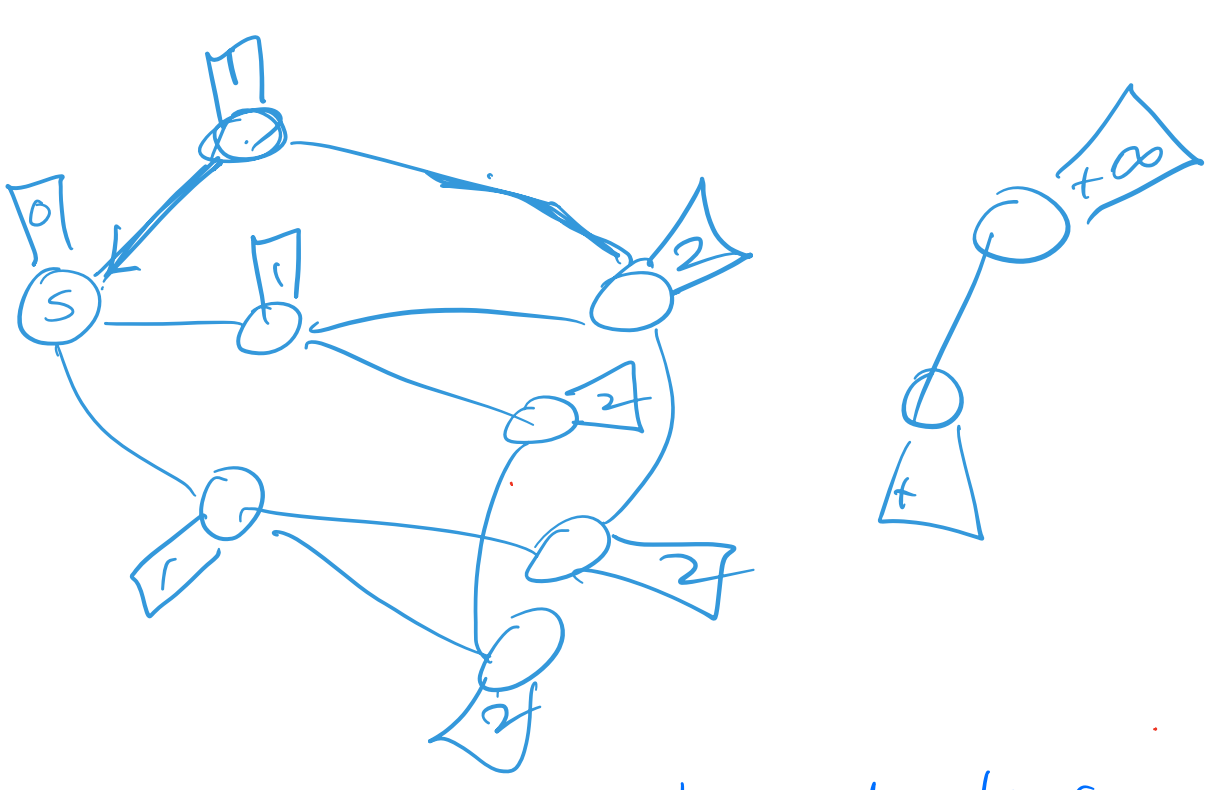
Shortest Paths – Unit length edges

With a couple of lines of added code, BFS can compute the length of each vertex's path from s (∞ if not in same connected component)

$$\ell(s) = 0; \quad \ell(v) = +\infty \quad \forall\, v \neq s. \leftarrow \Theta(n).$$

and, when marking w as visited...

$$\ell(w) = \ell(v) + 1 \leftarrow \Theta(n)$$



Question: what does this do to running time?

Question: What if the edges have distances...

i.e. not all edges have same value Θ = INT_MAX

...we will figure out an algorithm for this later...

Running time of Augmented BFS

$\Theta(n + m' + n)$ n' = number of vertices in connected component of s

$\Theta(n + m')$

m' = number of edges in connected component of s.

n = # edges in G.

Applications

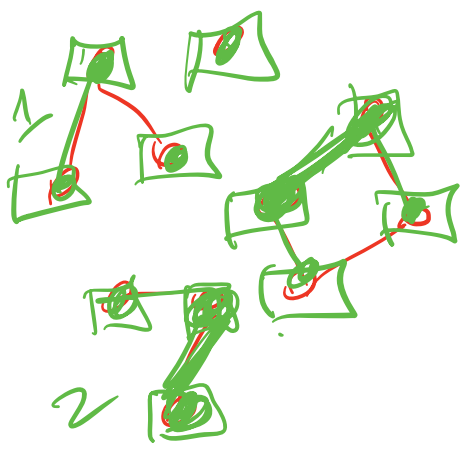
Detecting Network failures

Data Visualization

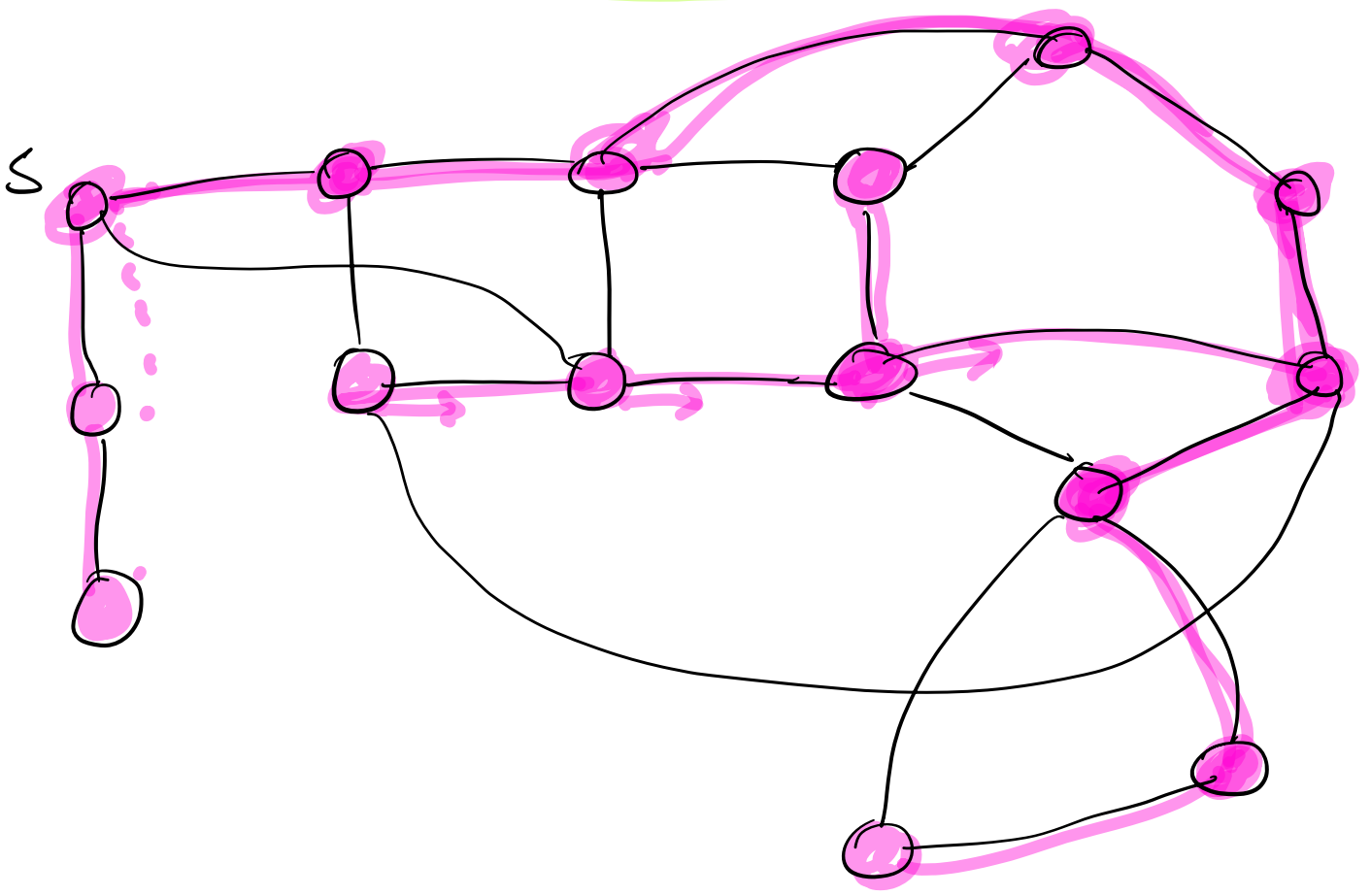
Clustering

How to compute number of connected components in a graph:

- #components = 0.
- While $\exists$ an unmarked vertex v
  - #components ++
  - BFS(v)

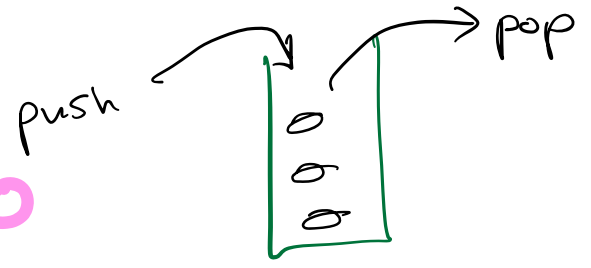


Depth-First Search (DFS)



Recall: Stack

Iterative version:



DFS

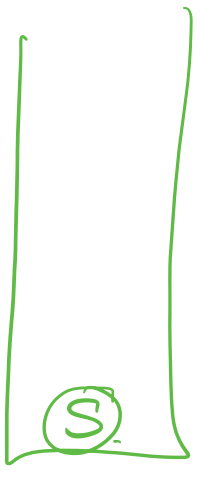
Input: $G=(V,E)$ in adj-list representation; vertex $s \in V$

Postcond: v is reachable from s iff marked[v] = true.

```
∀ v ∈ V   marked[v] = false

S.init(); S.insert(s)      // S is a stack.

while !S.empty()
    v = S.front(); S.pop();
    if !marked[v]
```

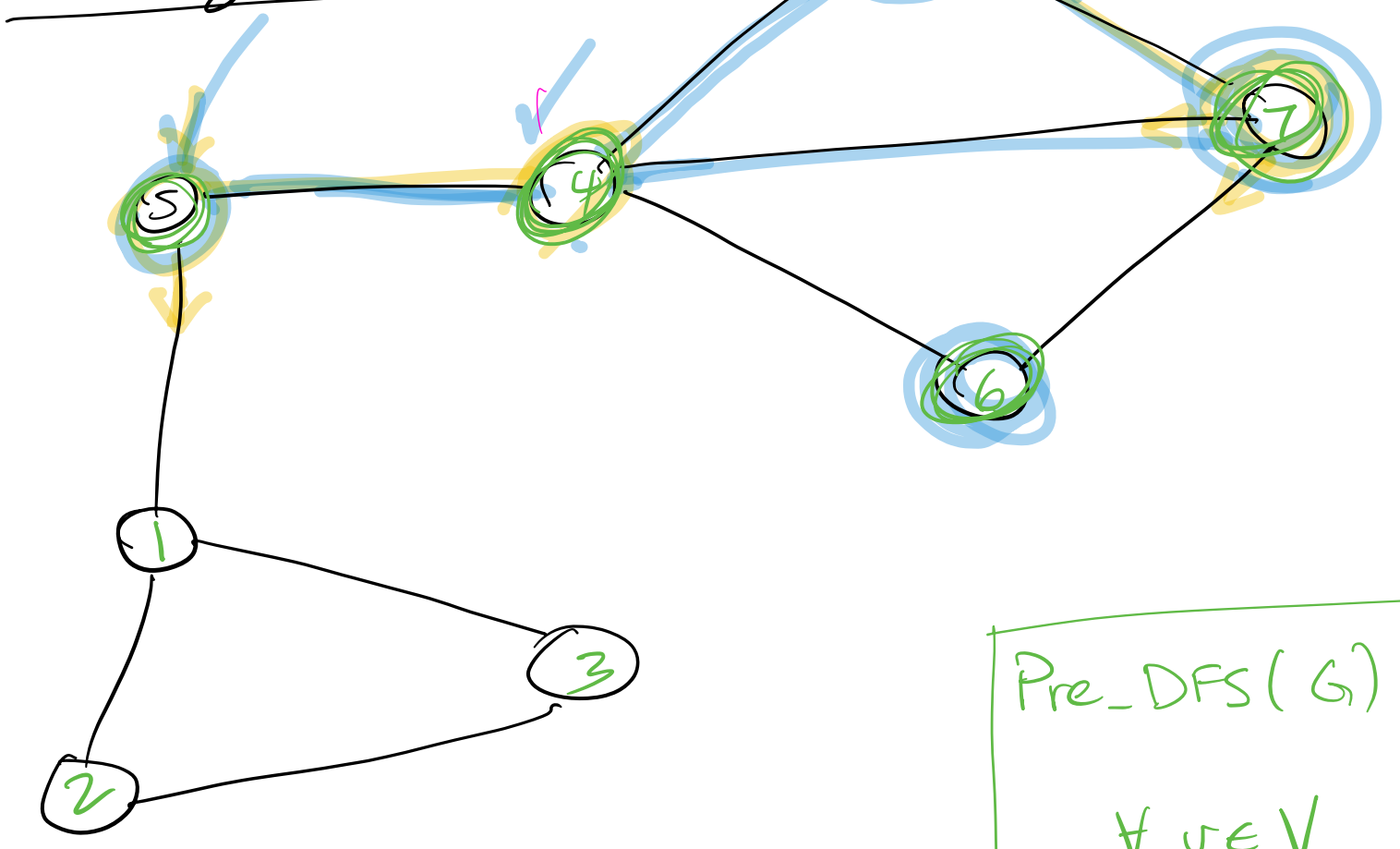


∀ edge (v, w)

S. push (w)

marked[w] = true

Running time



2

Recursive version

```
DFS(G, u)
    if marked[u] return
    marked[u] = true
    ∀ edge (u, v)
        if !marked[v]
            DFS(G, v)
```

```
Pre_DFS(G)
    ∀ v ∈ V
        marked[v] = false
```

DepthFirstSearch(G, s)

// Input : • graph $G=(V,E)$ in adj list representation
• $s \in V$

// Postcondition : $\forall v \in V$, v is reachable from s iff marked[v] == true.

$\forall v \in V$
    mark[v] = false

DFS(s)

Note: "marked" is global to all recursive calls. otherwise it won't work (or will need to be passed as a parameter).

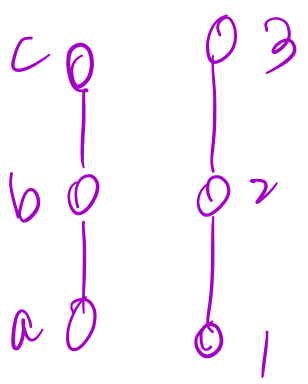
Correctness of DFS:

- it halts, because DFS will only be called $\leq$ once on each vertex
- it is correct, as it is a version of generic Search.
  (a proof by induction, on distance (in # of edges) will work even better for recursive DFS)

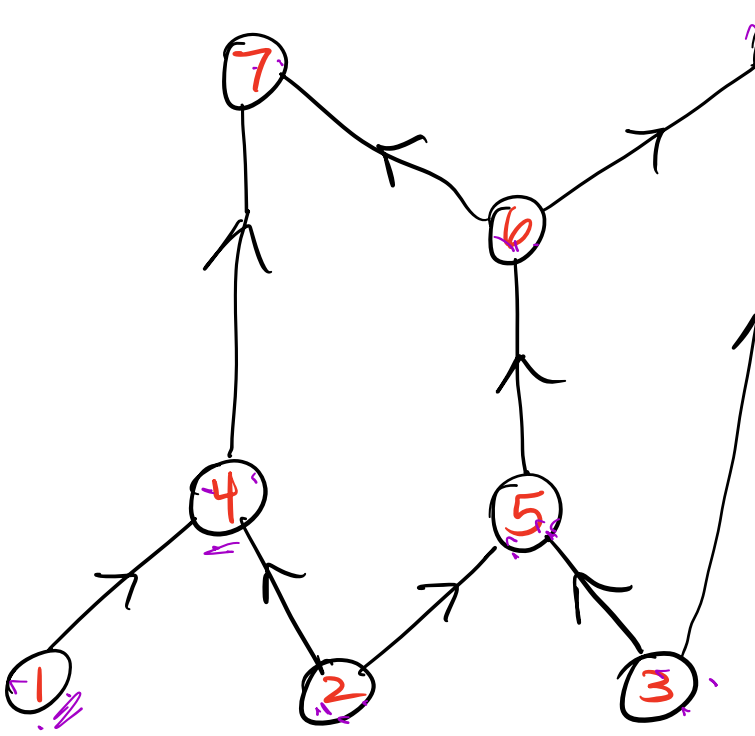
Applications for DFS

Topological Sort

- On directed Acyclic graphs: DAG
  each edge is directed; and ∄ directed cycles.



a | b 2 c 3



← can't do task 8 till have completed task 3.

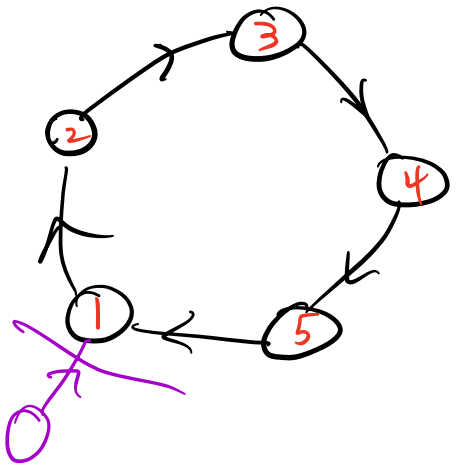
1 2 4 3 5 6 7 8 ← topological sort of the DAG

1 2 3 4 5 6 7 8

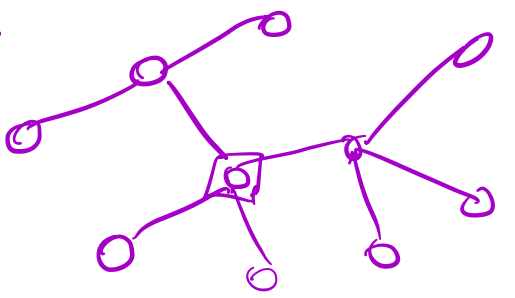
3 2 5 6 8 1 4 7

⋮

Can't find a topological sort $\iff$ $\exists$ directed cycle.



A graph that is a tree



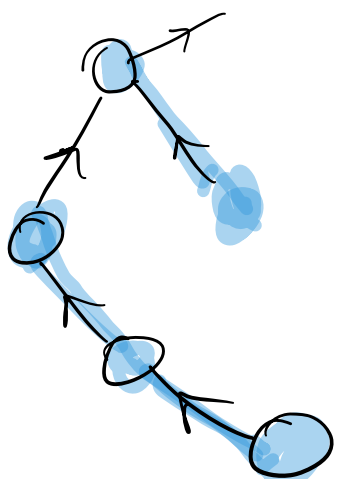
Directed Acyclic graph (DAG)

- not necessarily a tree (differs from undirected case)

$\forall$ DAG has a topological sort

$\forall$ DAG has ≥ 1 "source" (a vertex with no in-edges)

- Keep going backwards along edges
  - will either



```
Setup_DFSTopo (G)
    ∀ v ∈ V   marked[v] = false
    curLabel = |V|   ← n
    ∀ v ∈ V  do
        if !marked [v]
            DFSTopo (G, v)
```

DFSTopo (G, v)

Input: G = (V, E) in adj-list form; $v \in V$

postcond: f[1] f[2] ··· f[|V|] is a topo-sort

curr Label = |V| // = n

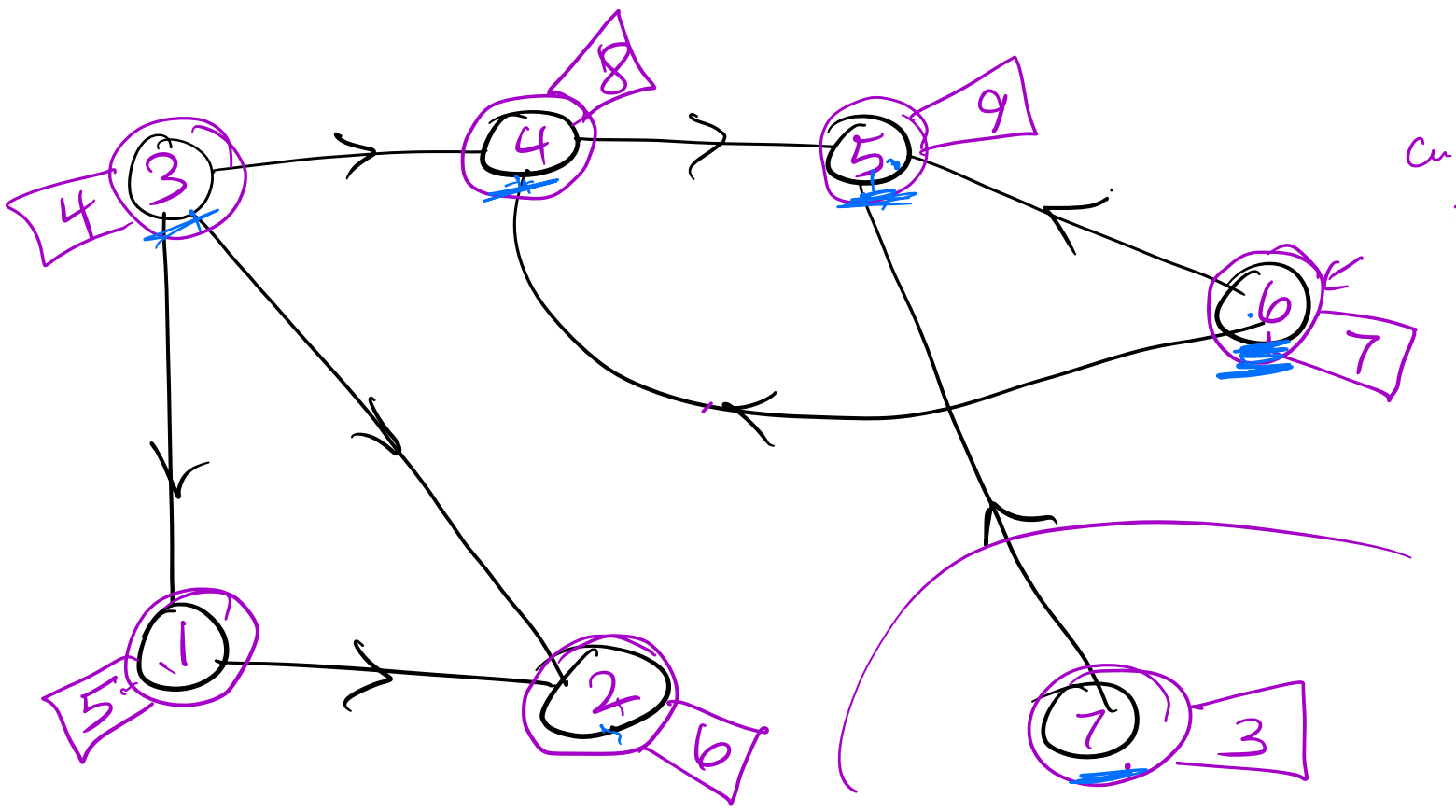
Marked [v] = true ✓

∀ edge (v, w)

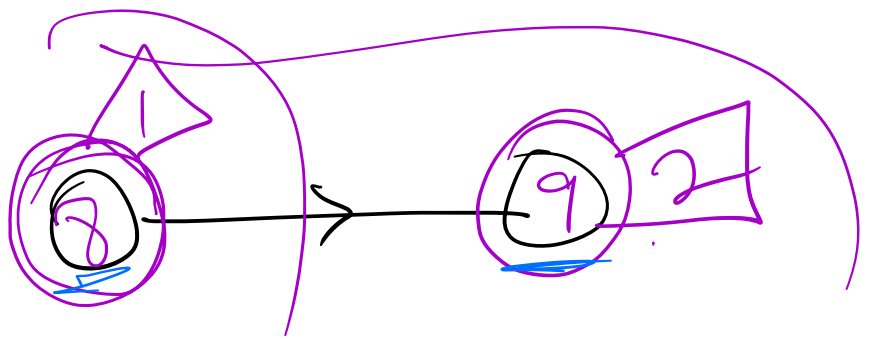
if ! marked [w]

DFS Topo (G, w)

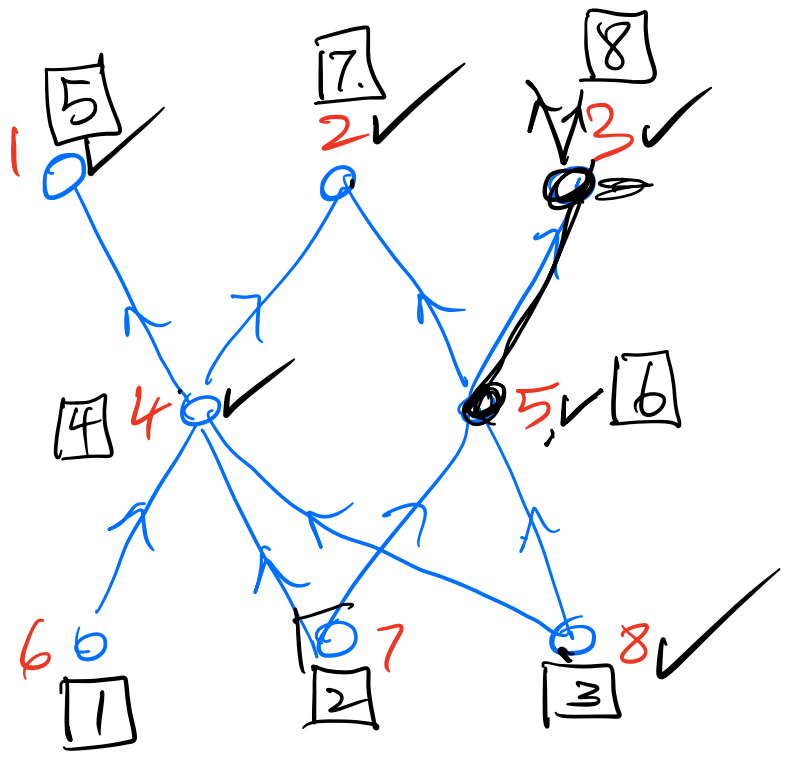
f[v] = cur Label --



curlabel
= 9
8
7
6
5
4
3
2



8 9 7 3 1 2 6 4 5



8 7 5 3 6 4 2 1

curr Label = 8 ~~7 6 5~~
4

6 7 8 4 1 5 2 3.