Depth First Search

- Graph $G = (V, E)$ directed or undirected
- Adjacency list representation
- Goal: Explore every vertex and every edge.

DFS(G)
for each $u \in V$
    do color[u] $\leftarrow$ white
time $\leftarrow 0$
for each $u \in V$
    do if color[u] = white
        then DFS-visit(u)

DFS-visit(u) .... Called once / vertex
color[u] $\leftarrow$ Gray
time $\leftarrow$ time + 1
d[u] $\leftarrow$ time .... (Discovered)
for each $v \in$ adj[u]
    do if color[v] = white
        then DFS-visit(v)
} edges of u explored
color[u] $\leftarrow$ Black
time $\leftarrow$ time + 1
f[u] $\leftarrow$ time .... (Finished)

$$\text{Time} = \Theta(V) + \Theta\left(\sum_u |\text{adj}[u]|\right) = \Theta(V) + \Theta(E) = \Theta(V+E)$$

- Be careful. DFS_vist is called once per vertex.
- The running time of DFS_vist (u) is $O(V)$ since it has to process edges of u.
- But total running time is not $O(V^2)$!

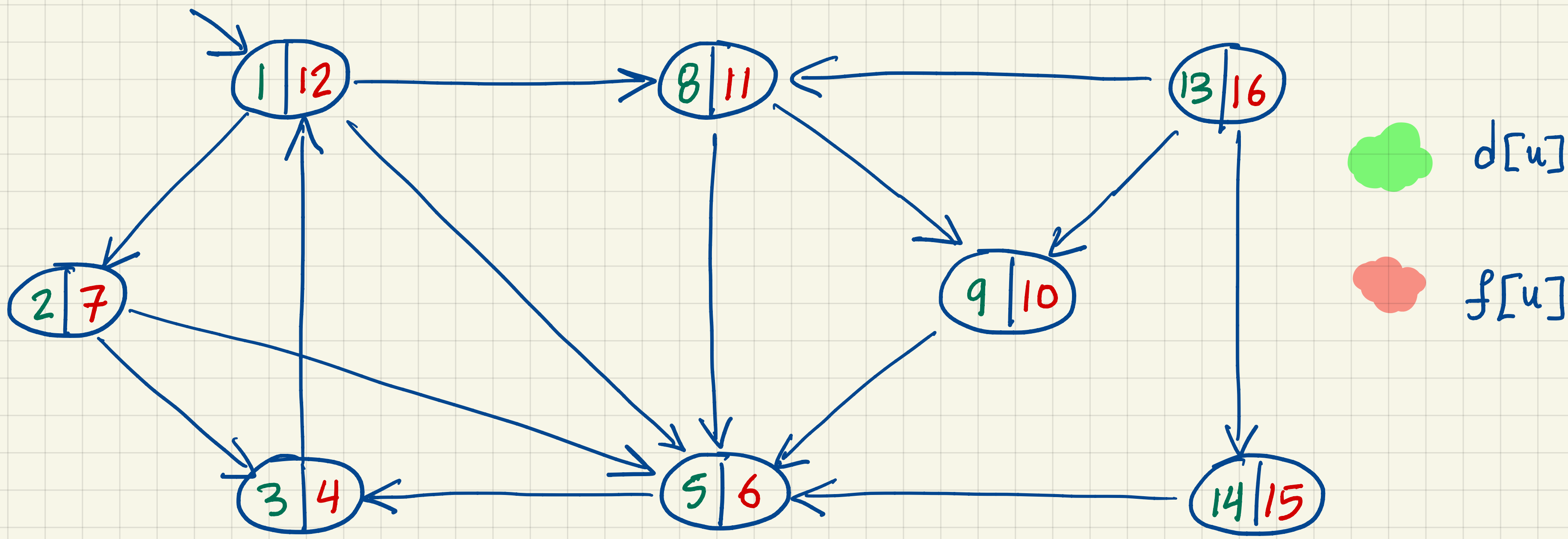
The processing of vertex u can be charged to edges.

Each edge charged once (or twice if undirected)

Rest of code charged to u.

$O(V) + O(E)$ aggregate analysis.

Example:



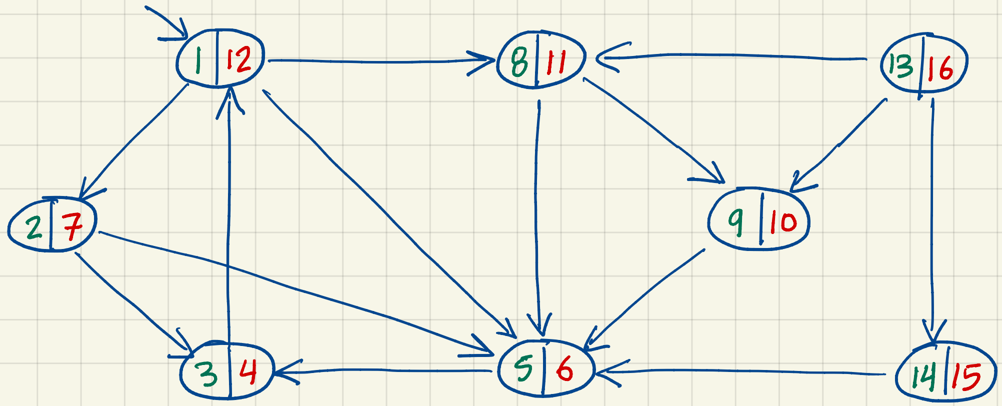
vertex u is gray from d[u] to f[u]

gray vertices $\Rightarrow$ stack of recursive calls

(started but not finished)

4 Kinds of edges

(u,v)



Tree edge: v encountered from u for first time. Gray → white

$$d[u] < d[v] < f[v] < f[u]$$

Back edge: From desendent to ancestor in the tree. Gray → Gray

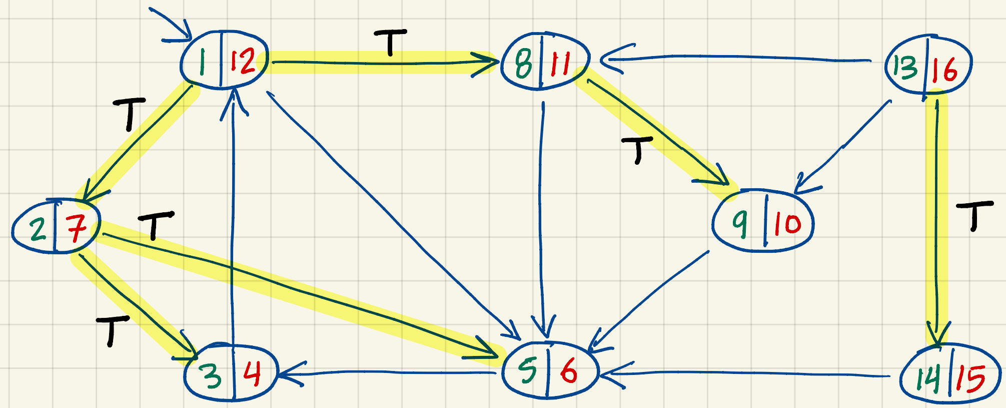
$$d[v] < d[u] < f[u] < f[v]$$

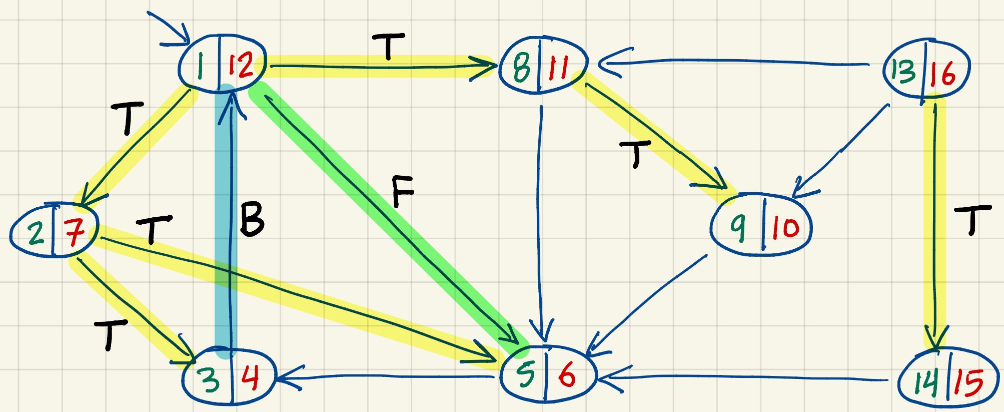
Forward edge: From ancestor to descentent in the tree Gray → Black

(same as tree edge)

Cross edge: All others. Gray → Black

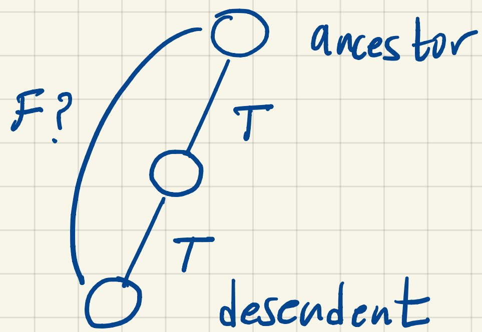
$$d[v] < f[v] < d[u] < f[u]$$

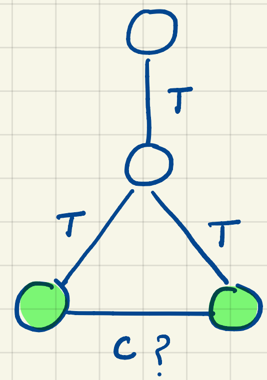

1 | 12
8 | 11
13 | 16
2 | 7
9 | 10
3 | 4
5 | 6
14 | 15
T
T
T
T
T
T
T


1 | 12
8 | 11
13 | 16
2 | 7
9 | 10
3 | 4
5 | 6
14 | 15
T
T
T
T
T
T
T
B
F

Useful facts :

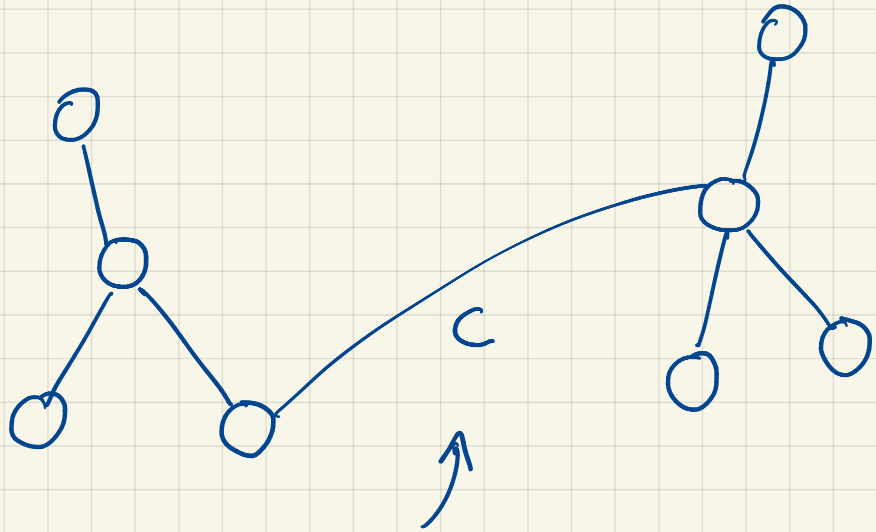
G is undirected $\Longrightarrow$ DFS produces only Tree & Back edges.





Assume $\exists$ Forward edge. But F? must actually be B because we must finish processing bottom vertex before resuming top

One of the two vertices must have been discovered first, making C? actually T. In fact labelling T/\T can't be right



should be T

Undirected G is acyclic $\Longleftrightarrow$ DFS yields no Back edges

Acyclic $\Longrightarrow$ No Back edge since back edge means cycle

No Back edge $\Longrightarrow$ only T edges (from before) so graph is forest.

Check for cycles in undirected graph in $O(V)$ time

Run DFS

- if encounter a B edge $\Rightarrow$ $\exists$ cycles
- No need to explore more than $|V|$ edges

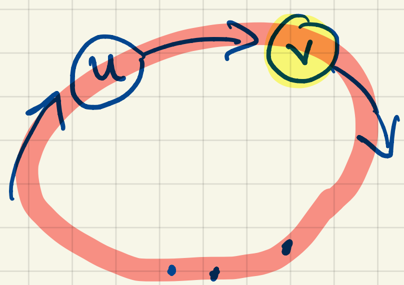
What about directed graphs? We can still claim:

Directed G is acyclic $\Longleftrightarrow$ DFS yields no Back edges

- acyclic $\Rightarrow$ (as before) No back edges since Back edge means cycle.

- No Back edge $\Rightarrow$ no cycle.

Proof by contradiction: Assume $\exists$ cycle and let v have smallest $d[v]$ on cycle.



all vertices on cycle are white when v discovered $\Rightarrow$ will visit all before returning from DFS-Visit(v). Therefore (u,v) is Back edge, contradiction.

$O(V+E)$ time to determine if cyclic or not

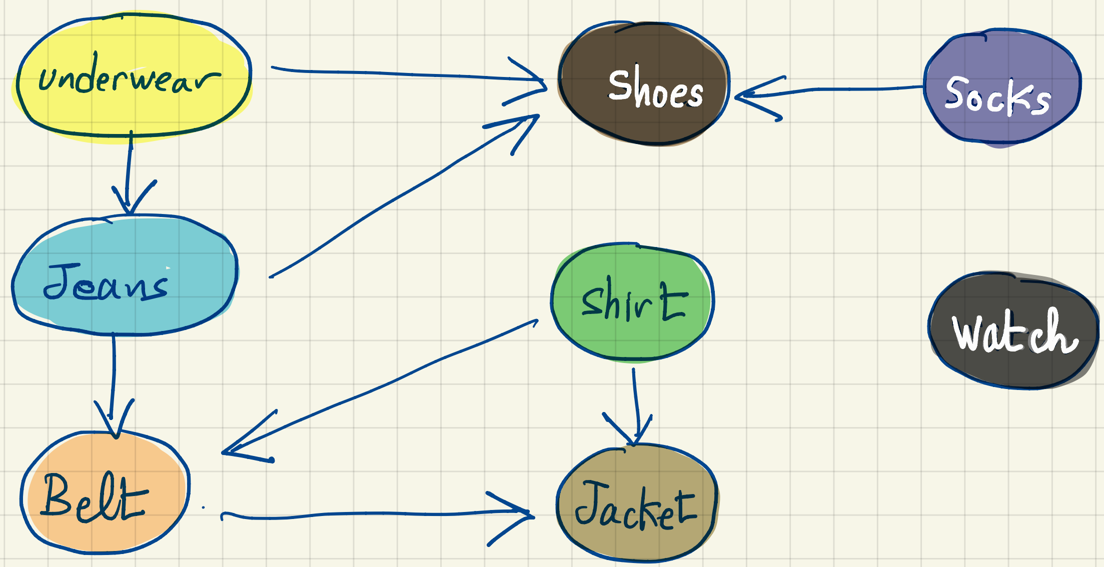
Topological Sorting of DAG

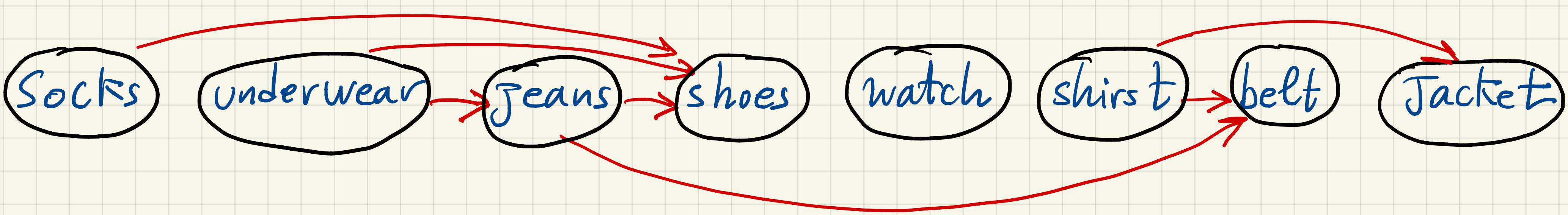
Order vertices of DAG such that

$$(u,v) \in E \implies u \prec v$$

e.g. order tasks that depend on each others.

Example:





Topological_Sort (G)

run DFS $\qquad$ $O(V+E)$ time

when vertex finished, output it

<u>Claim:</u> Vertices output in reverse topological order

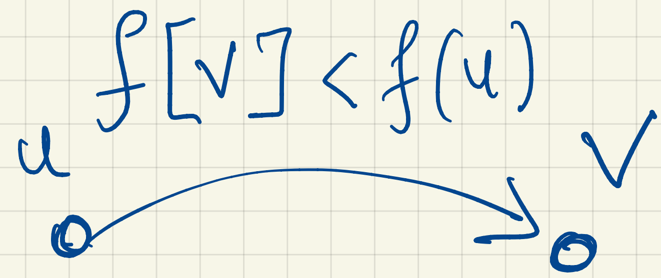
$$(u,v) \in E \Rightarrow f[v] < f[u]$$

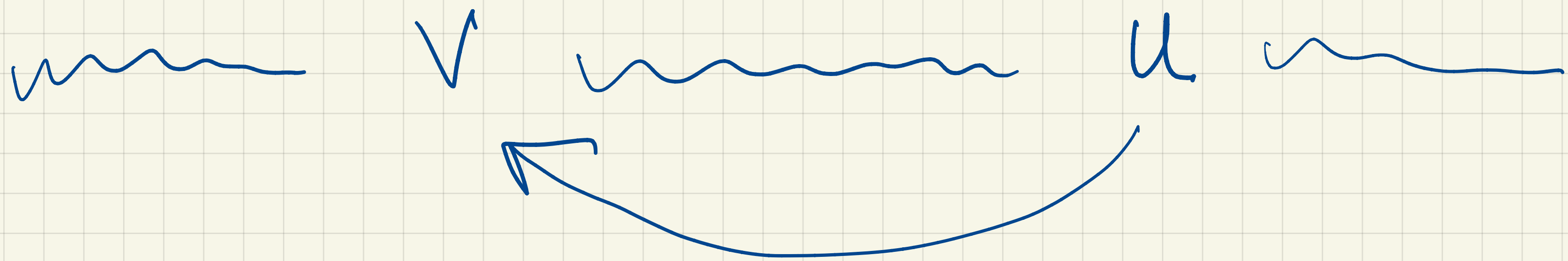
Proof: When (u,v) is explored, u is gray

v gray : (u,v) back edge, contradiction

v white : v discovered, v has to finish before coming back to u, so $f[v] < f[u]$

v black : v already finished, so $f[v] < f[u]$





<u>Note:</u> Single source shortest path can be found in linear time in DAGs.

- Topological sort
- Relax edges of vertices in topological order.