Date: 05/05/21

Black Board

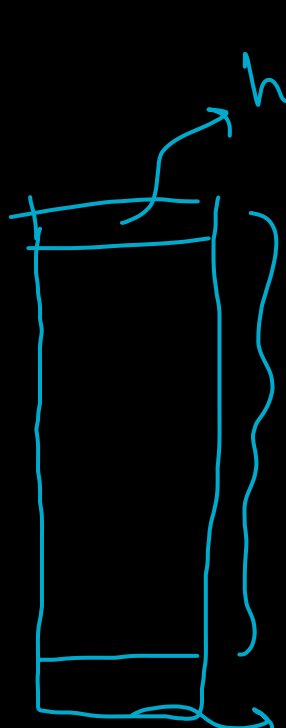
Design and Analysis of Algorithms

Topics:

- Graph Algorithms II
  - Enhanced DFS ✓
  - DAG Detection ✓
  - Linearizing a DAG ✓

explore code

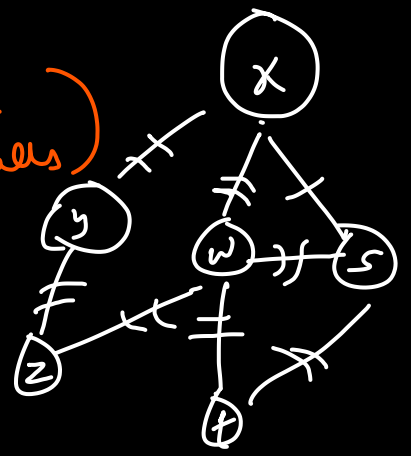
```
Explore (G=(V, E), x)                    // S is a global stack
    visited[x] ← true
    /* Pre-processing */
    pre[x] ← clock
    clock ← clock+1
    For each (x, y) ∈ E
        IF visited[y] != true
            Explore(G, y)
    /* Post-processing */
    post[x] ← clock
    clock ← clock
    S.push(x)  // O(1)
```



|V| # of explore } DFS takes linear Running Ti°

→ Sum up all the work done over |V| explore calls.

↳ { total # of explore (across all Explore calls)
loop iterations : 2|E|



$T(G(V,E)) = O(|V| + |E|)$

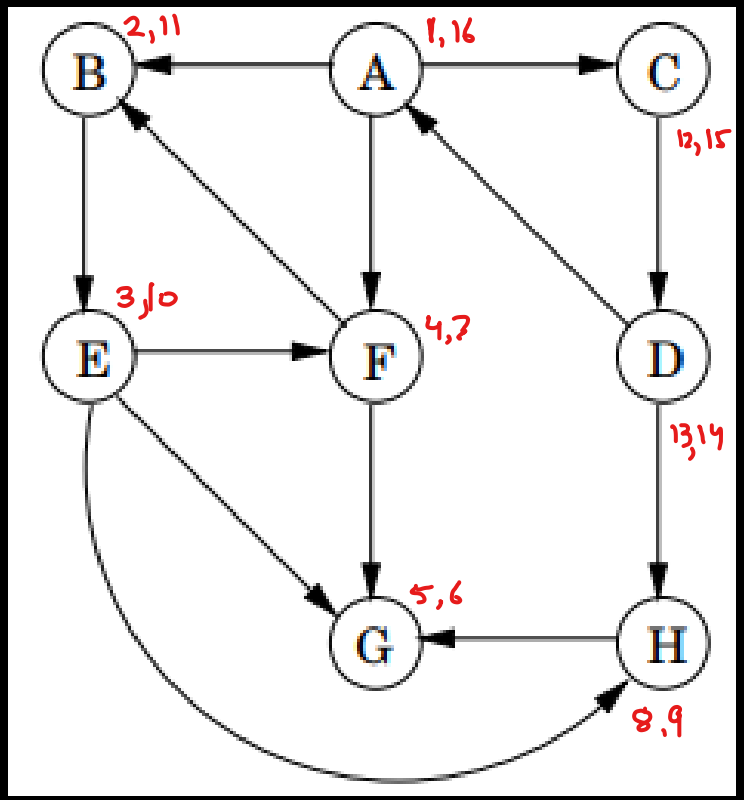
linear.

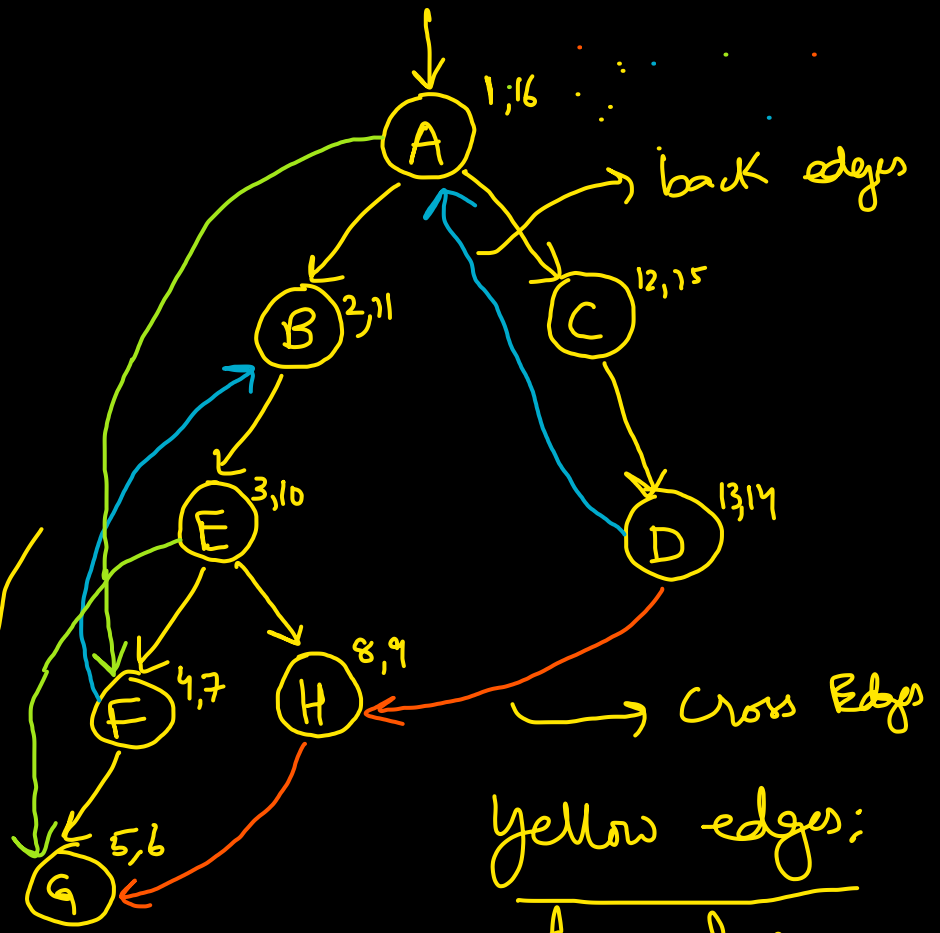
dfs code

```
dfs(G=(V,E))

    For each x∈V:
        visited[x] = false
    clock←1                          ——→ s.clear()
    For each x∈V:
        IF visited[x] != true
            Explore(G,x)
```

DFS Pre/Post Numbers





$Pre(A) < Pre(D) < Post(D) < post(A)$ forward edges.

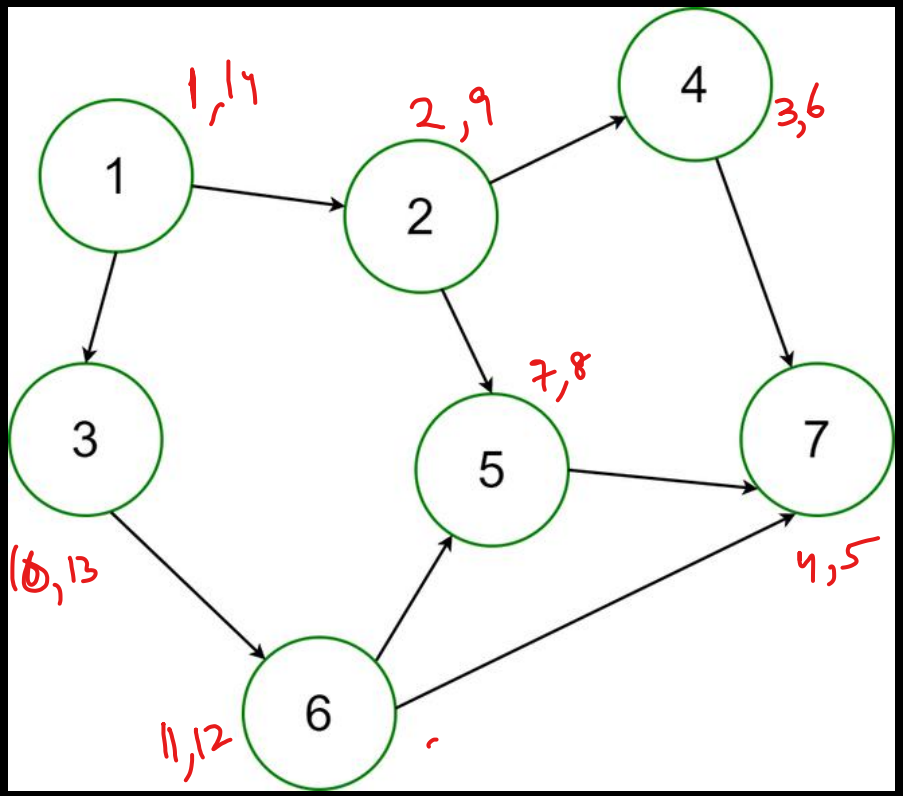
$\Rightarrow (D,A)$ is a back edge $\Rightarrow$ Graph not A Graph

DFS Edge Types

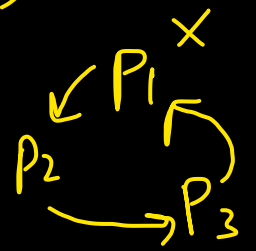
- Tree Edge
  - From a parent node to a child node
- Backward Edge
  - From a descendent to an ancestor.
- Forward Edge
  - From an non-parent ancestor of x to x.
- Cross Edge
  - From a non-ancestor of x to x.

Directed Acyclic Graphs and their importance

1, 2, 3, 7, 9 → n!



(DAG) =



A few example applications

- Scheduling
- Data Processing Networks
- Version Control Systems
- Citation Networks
- Crypto-currency (the dagger algorithm)
- Data Compression
- Component Analysis

1, 3, 6, 2, 5, 4, 7 → topological order.

$T(G=(V,E)) = O(|V| + |E|)$

Algorithm to detect a DAG

→ Perform DFS and mark every node with pre/post numbers. ⟶ $O(|V|+|E|)$

→ Go through all edges. → $O(|E|)$

If $\exists\ e \in E,\ e=(x,y)$

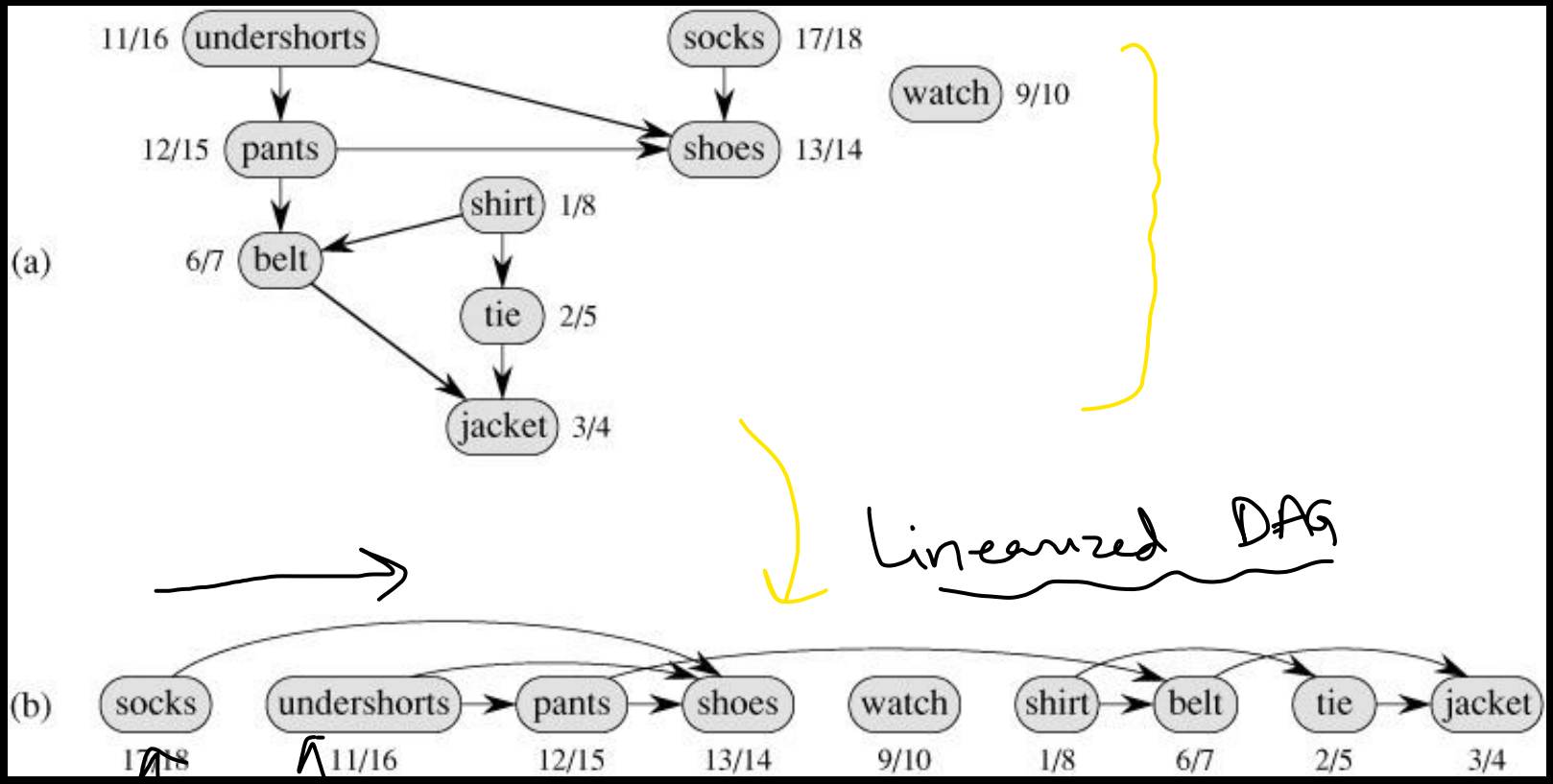
such that $pre(y) < pre(x) < post(x) < post(y)$

⇒ backward edge exists

⇒ G is not a DAG.

→ Report G as DAG

Linearizing a DAG



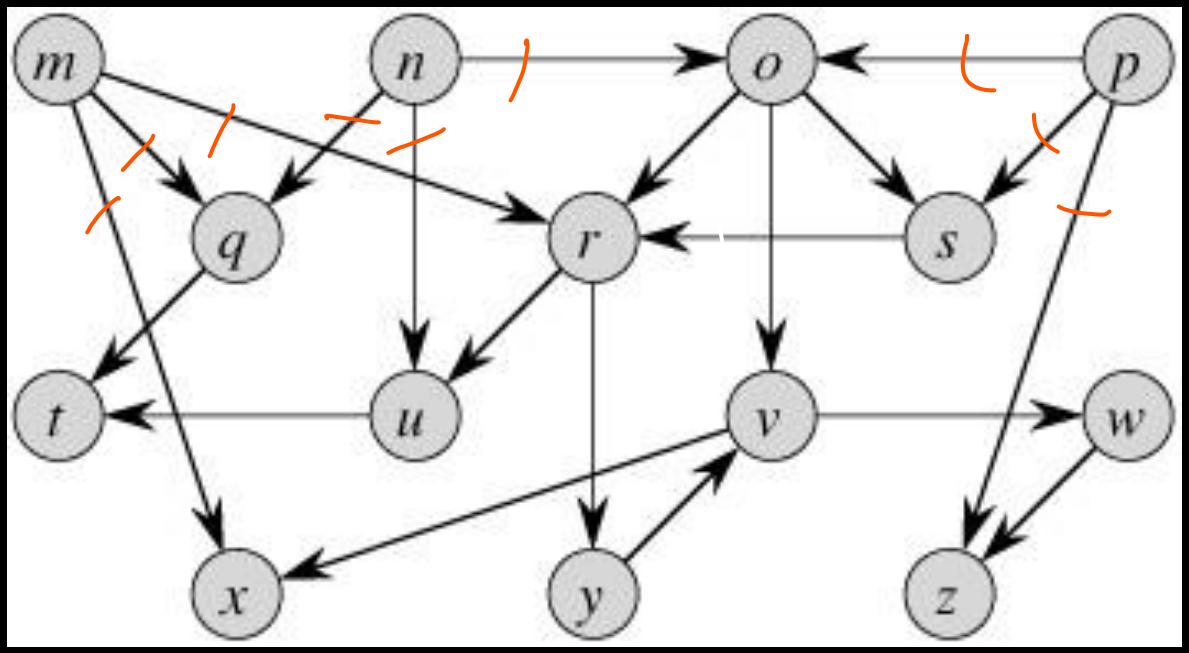
Linearizing a DAG Algorithm I

Pick vertices source by source

$O(|V|^2+|E|)$

Naive Time

$O(|V|^2)$ → $O(|V|\lg|V|)$



$m = p = q = \cdots = n$

$O(|V|\lg|V|+|E|)$

goal

vertices with indegree 0: source

" " outdegree 0: sink

$O(|V|+|E|)$

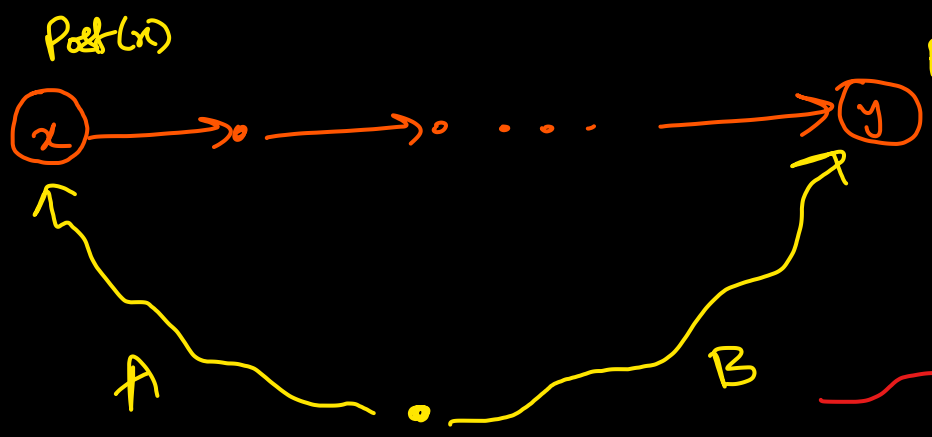
Linearizing a DAG Algorithm II

Topological Sort

O(|V|+|E|)

The Insight:

we can use the post Numbers.



x , y

not imp.

⇒ If we enter x before y, post(y) < post(x)

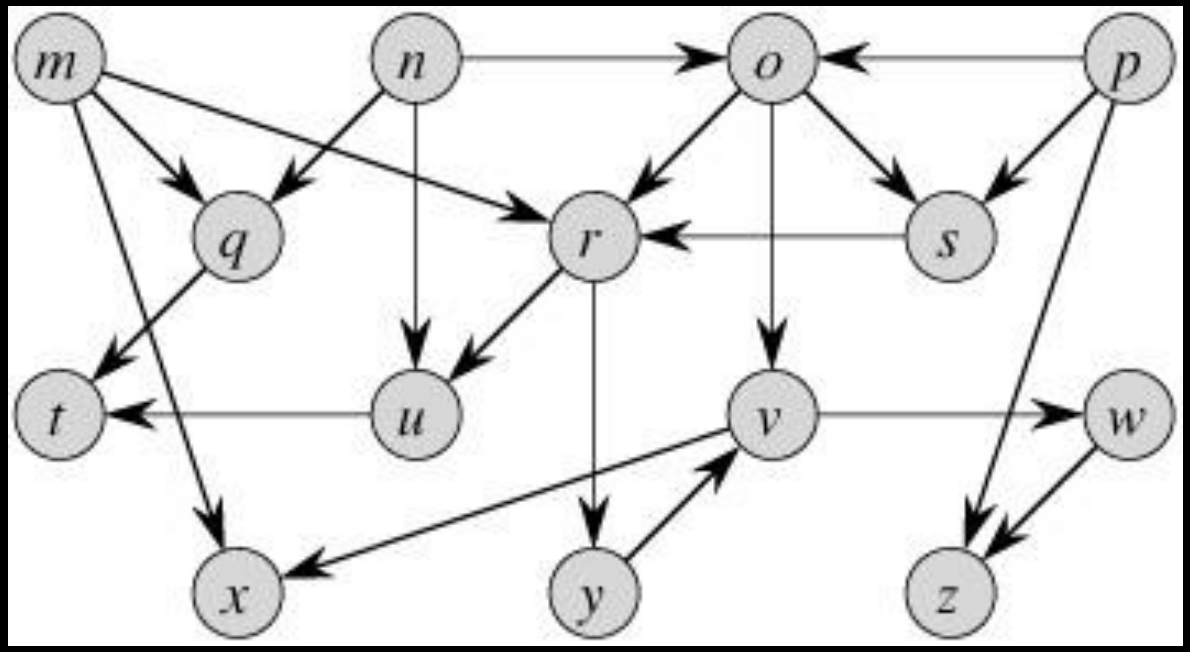
⇒ If we enter y before x, post(y) < post(x

Prec(x) < post(y)

post(y) < post(x

Linearizing a DAG Algorithm II
Topological Sort

- The Algorithm:



Running Time of Topological Sort?