Design and Analysis of Algorithms

L16: Topological Sorting

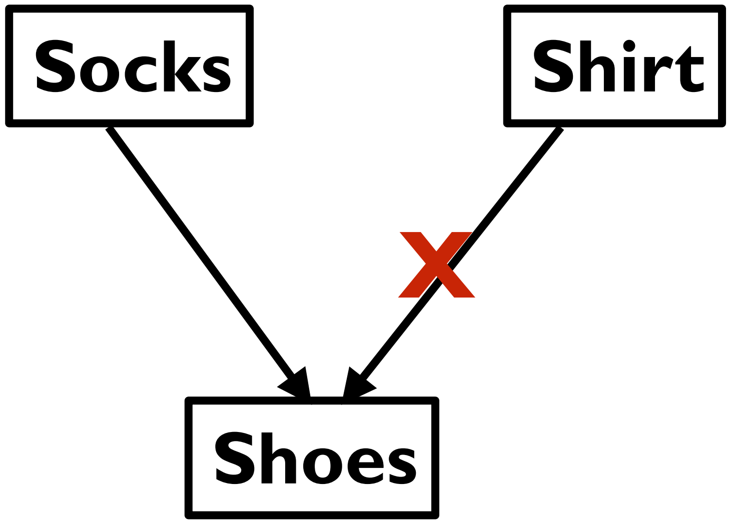
Dr. Ram P Rustagi
Sem IV (2020-Even)
Dept of CSE, KSIT
rprustagi@ksit.edu.in

Resources

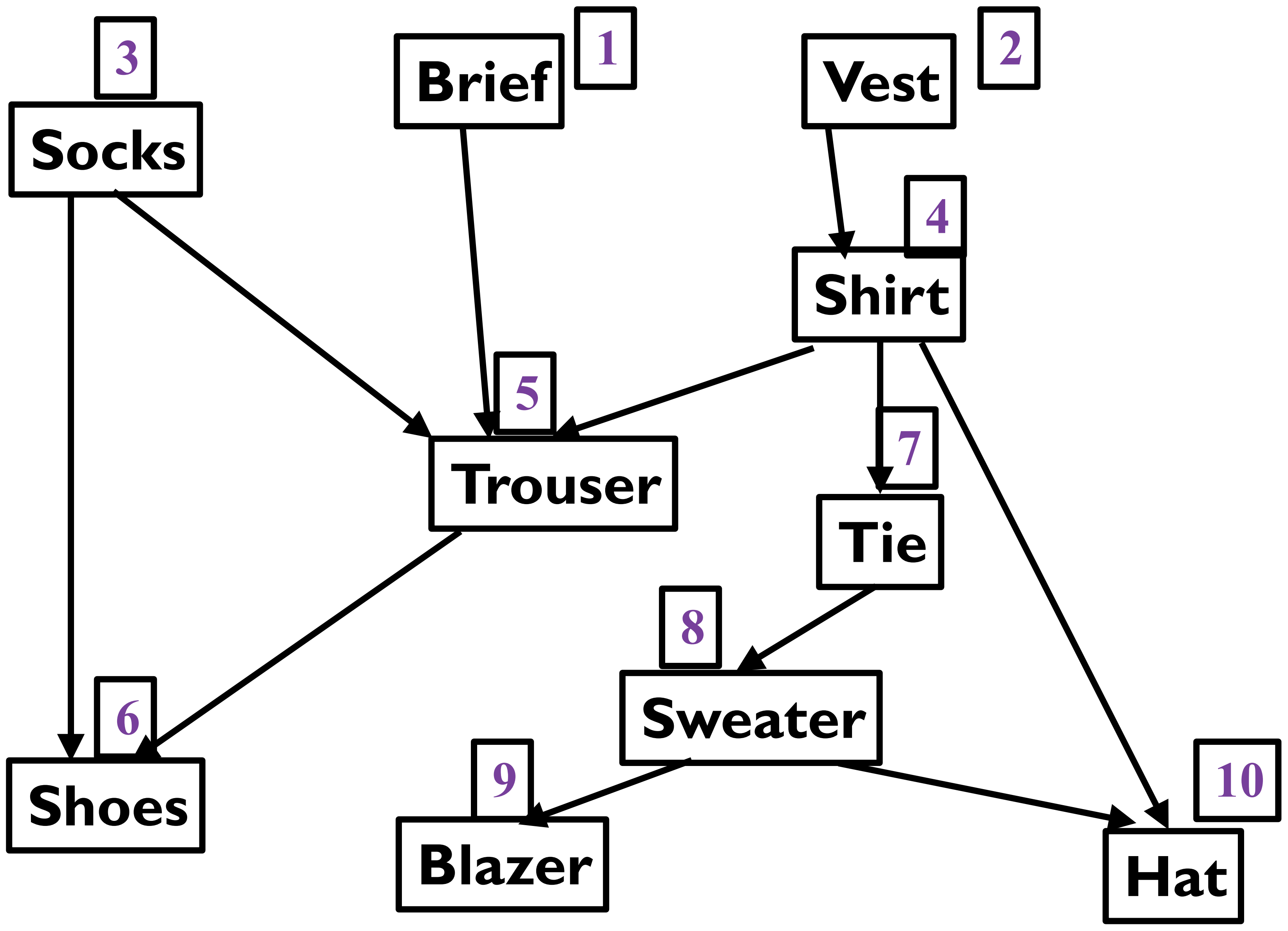
- T1: Sec 5.1-5.3 - Levitin
- R1: Introduction to Algorithms
    - Cormen et al.
- Introduction to Algorithms - A creative approach
    - Udi Manber

Topological Sort Example

- Show the dependency graph in the order of wearing man's cloths
  - Blazer (Coat)
  - Brief
  - Hat
  - Shirt (tucked-in)
  - Sweater
  - Tie
  - Trouser
  - Socks
  - Shoes
  - Vest



Topological Sort Example



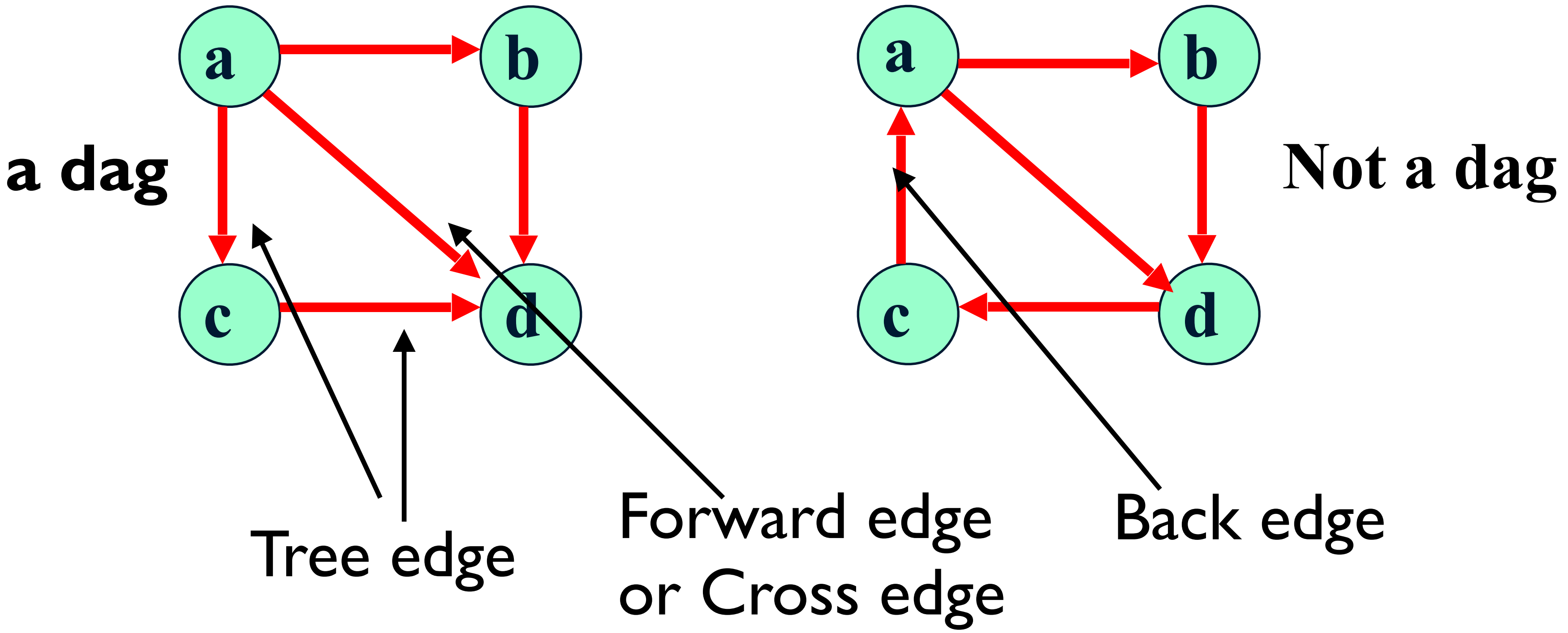
Topological Sort Example

- Show the dependency graph in the order of wearing man's cloths.
  - Belt (new). Define dependency
  - Blazer (Coat)
  - Brief
  - Hat
  - Shirt (tucked-in)
  - Sweater
  - Tie
  - Trouser
  - Socks
  - Shoes
  - Vest

Directed Acyclic Graph (*dag*)

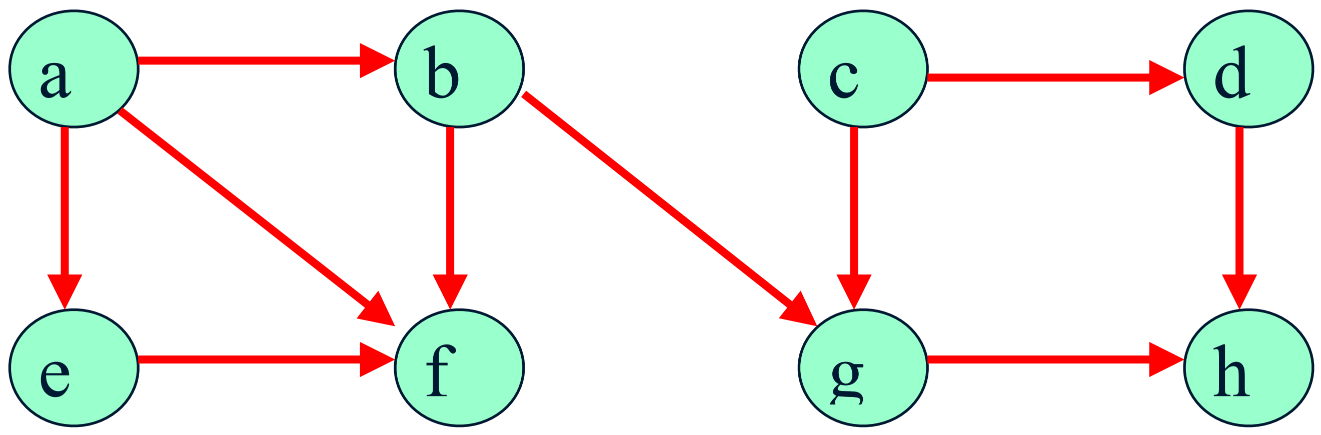
- *dag*: A directed graph with no (direct) cycles
- Useful in cases where pre-requisite constraints exists that define some dependency
- Topological sorting:
  - Ordering of vertices such that for every (directed) edge, the starting vertex of the edge is listed before the ending vertex.
    - pre-requisite courses for higher order courses
    - version control
  - Being a *dag* is a necessary condition for topological sorting to be possible.

Examples: *dag* and non-*dag*



Topological Sort: DFS Based

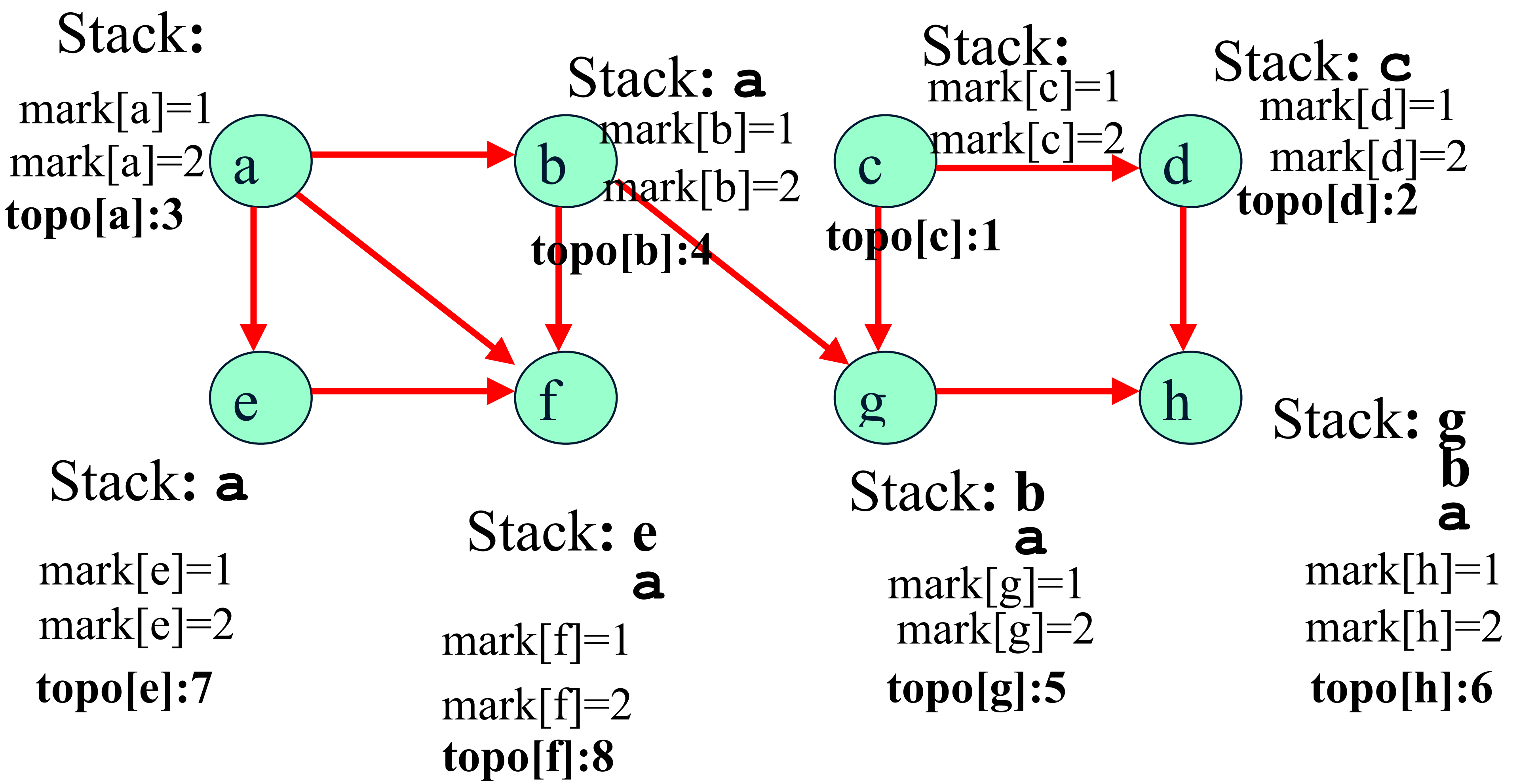
- <u>DFS-based algorithm for topological sorting</u>
  - Perform DFS traversal,
    - Note down the order vertices are popped off the traversal stack
  - Reverse order solves topological sorting problem
  - Back edges encountered? → NOT a dag!



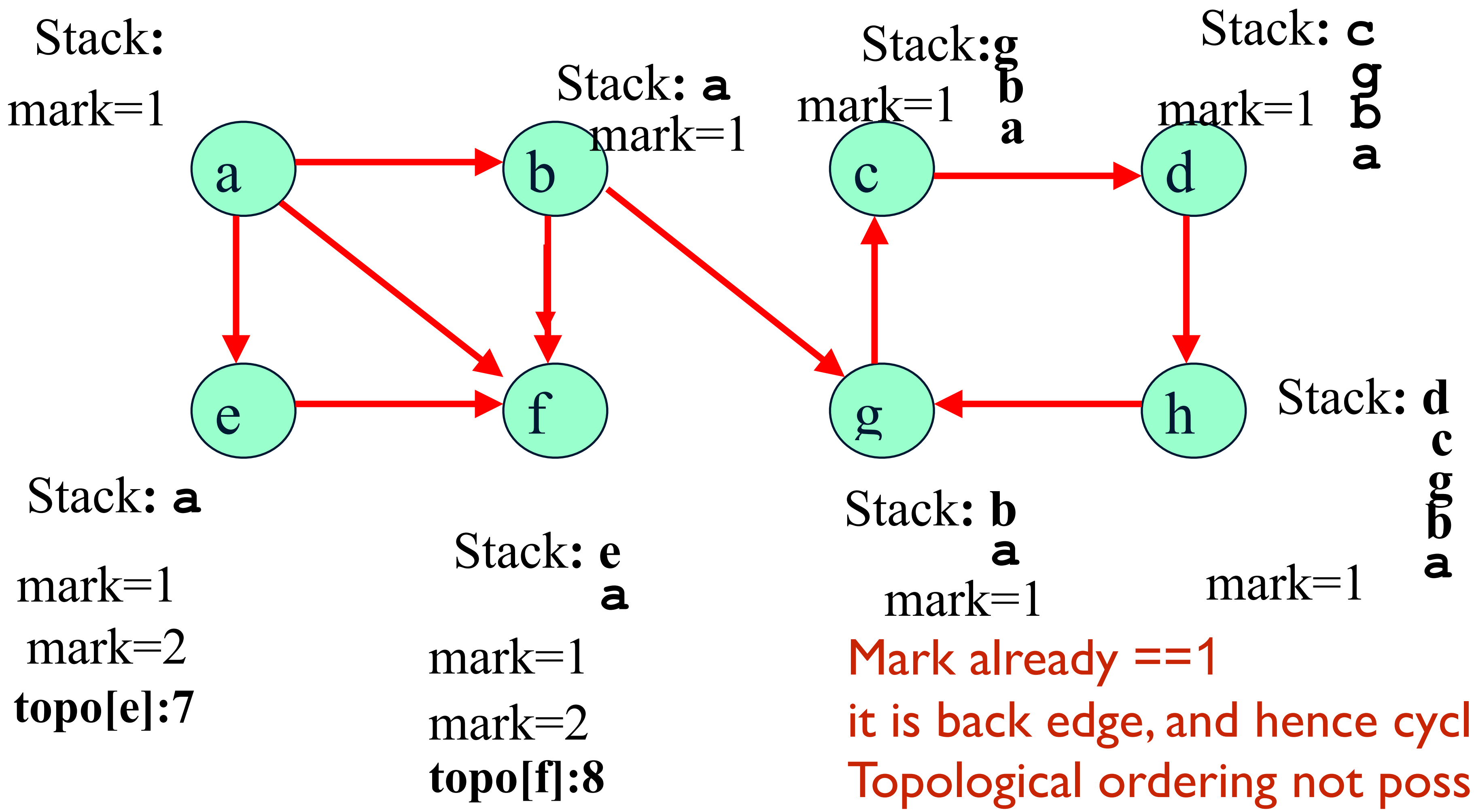
DFS Algo - Topo Sort

```
proc  DFS(v)
    mark[v]← 1  /* visiting
    for each vertex  w ∈ adjacency(v)  do
        if mark[w]  == 0  /* explore unvisited vertex */
            DFS(w)   /* node v is pushed on stack */
        if mark[w]  == 1 /* a back edge, and hence cycle */
            exit("graph has cycle")
    mark[v] ← 2  /* v is popped from stack, mark it visited */
    topo[v] = order--
/* initialization *
order ← N  /* reverse ordering */
for each vertex  v∈V  do
    mark[v]←0  /* unvisited */;
    topo[v]=0 /* order */
for each vertex  v∈V where v has no incident edges do
    DFS(v)   /* start from a some root */
```

DFS Based Topological Sort-dag



DFS Based Topological Sort-nondag



Mark already ==1
it is back edge, and hence cycle
Topological ordering not possible

Analysis: DFS based Topo Sort

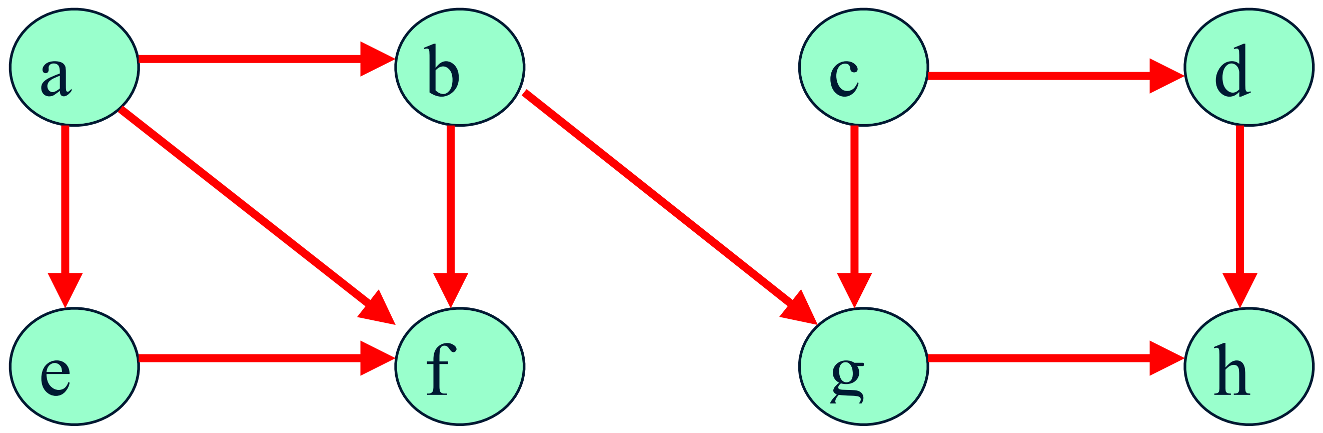
- When a vertex `v` is popped off the stack
  - no vertex `u` with an edge `(u,v)` can be among vertices popped off the stack before `v`.
    - Otherwise, `(u,v)` would be a back edge
    - Thus, any such vertex `u` will be listed after `v`.
- Time complexity
  - Same as that of DFS algorithm
    - `O(|V|+|E|)`

Topo Sort: Decrease and Conquer

- Given the diagraph (dag)
  - Identify a source (vertex with no incoming edges)
    - Remove this source from the diagraph
    - If there are multiple such sources
      - Take any one at random
    - Repeat the process in remaining diagraph
- The order in which vertices are removed
  - Provides a solution to the topological sorting
- If no source (without any incoming edges) is found
  - Then solution does not exists
  - There is a cycle and topological sorting can't be done

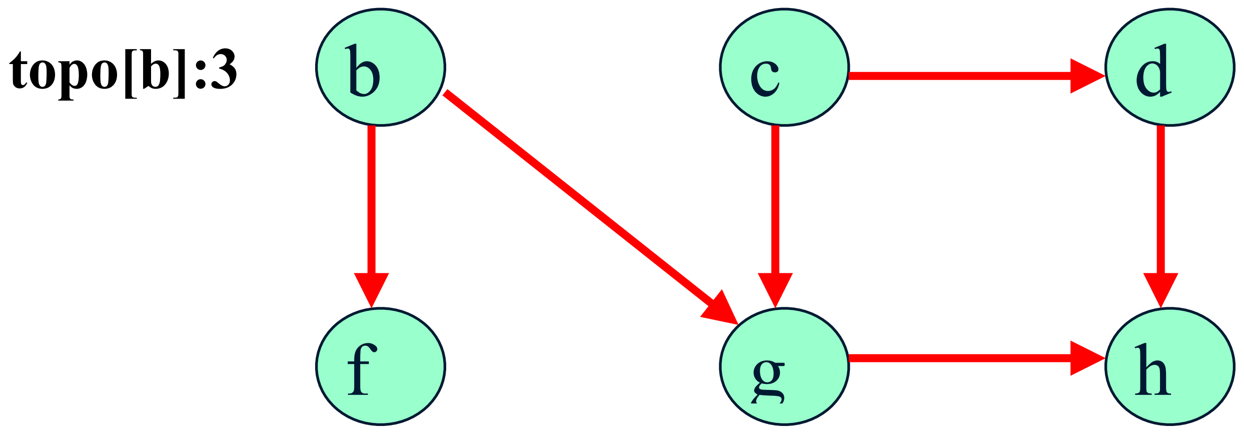
Topo sort (D&C)-`dag`

topo[a]:1



topo[e]:2

Topo sort (D&C)-`dag`



Topo sort (D&C)-`dag`

topo[f]:6

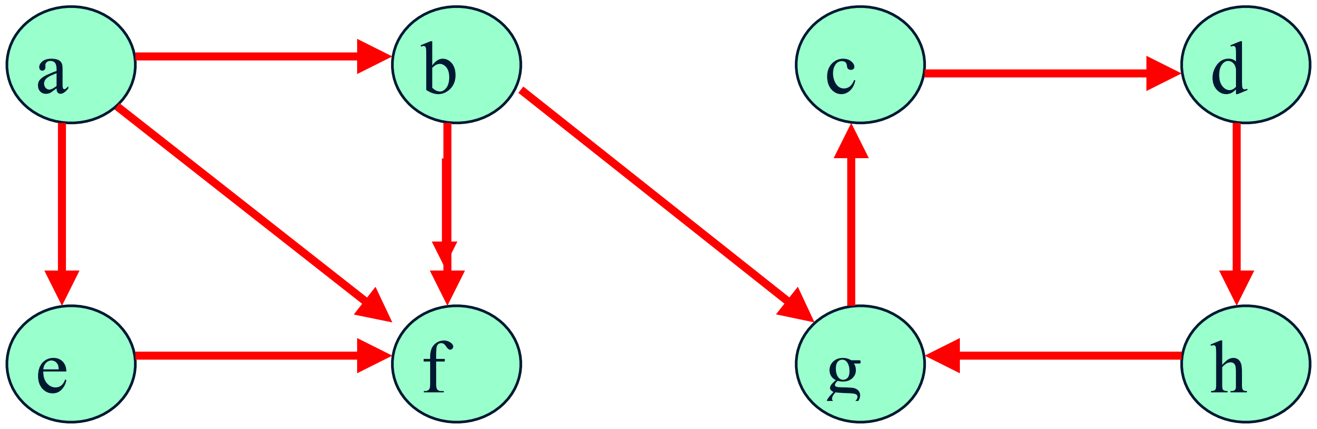
topo[g]:7

topo[h]:8

Topological sorting of vertices
a:1, e:2, b:3, c:4, d:5, f:6, g:7, h:8

Topo sort (D&C)-cyclic graph

topo[a]:1



topo[e]:2

Topo sort (D&C)-cyclic graph

topo[b]:3

topo[f]:4

Topo sort (D&C)-cyclic graph

No source can be found
i.e. there is no vertex with
no incoming edges.

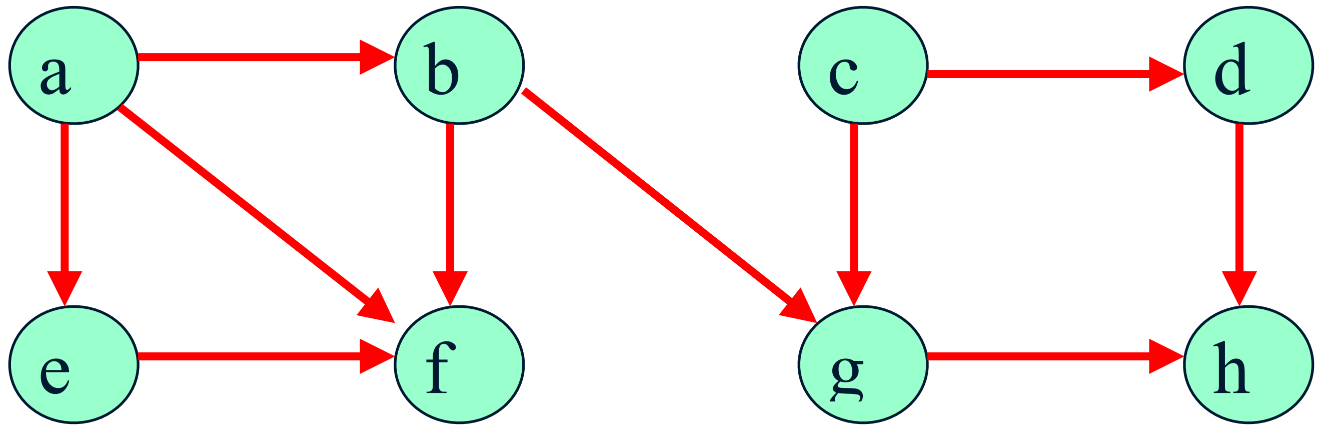
Thus, given graph is cyclic

Topo sort (D&C)-Algo

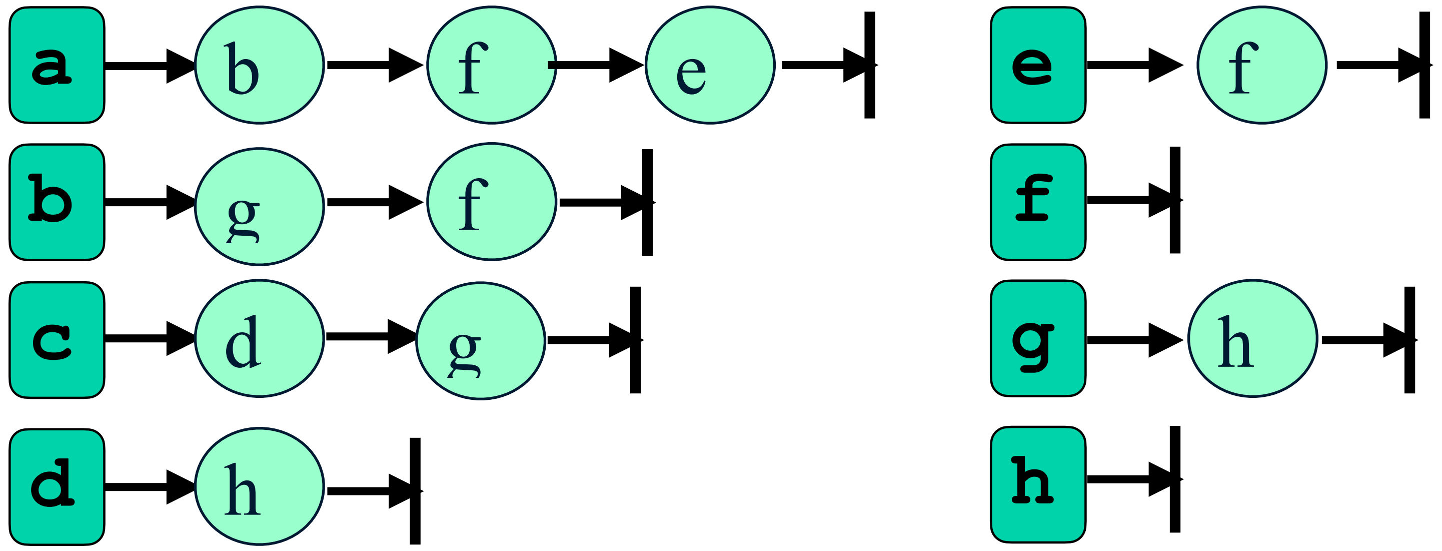
- Algo: `toposortdc(v,G)`
  - i/p: v is with in degree 0; o/p: topo order of v

```
topo[v]=order++
if nodes in G == 1
    return
For each edge v→w in G
    remove v→w from G
Remove v from G
Find a vertex w∈G such that in degree[w] is 0
    /* if no such vertex w, then graph is cyclic
    toposortdc(w,G)
/* main */
order=1
find v∈G such that indegree of v is 0
    toposortdc(v,G)
```

Topo sort (D&C)-Implementation



Adjacency List



Indegrees

a:0

bb01

c:0

dd01

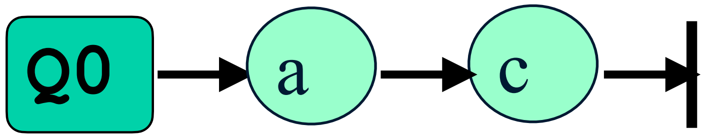
ee01

ff01f:2f:3

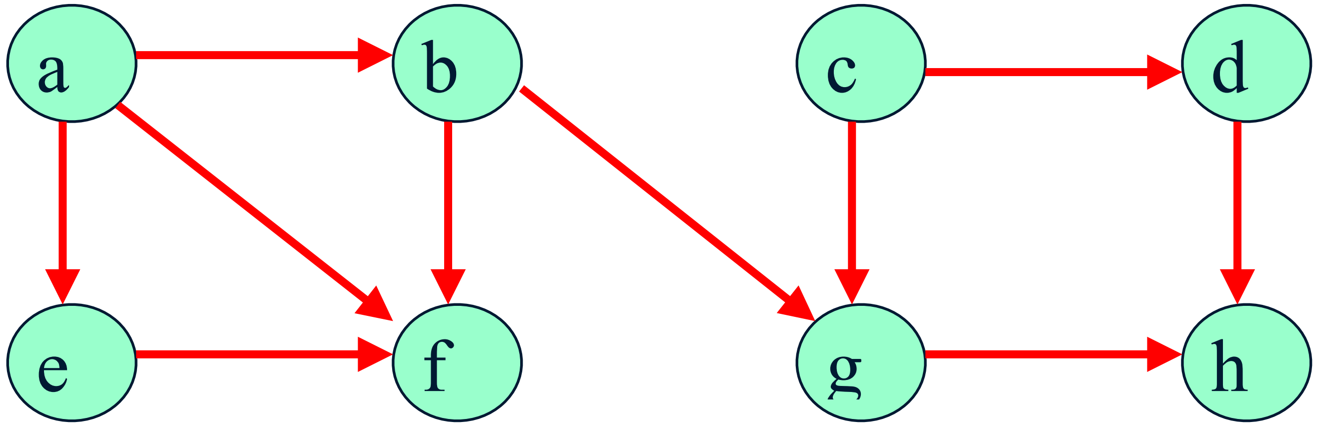
gg01g:2

hh01 h:2

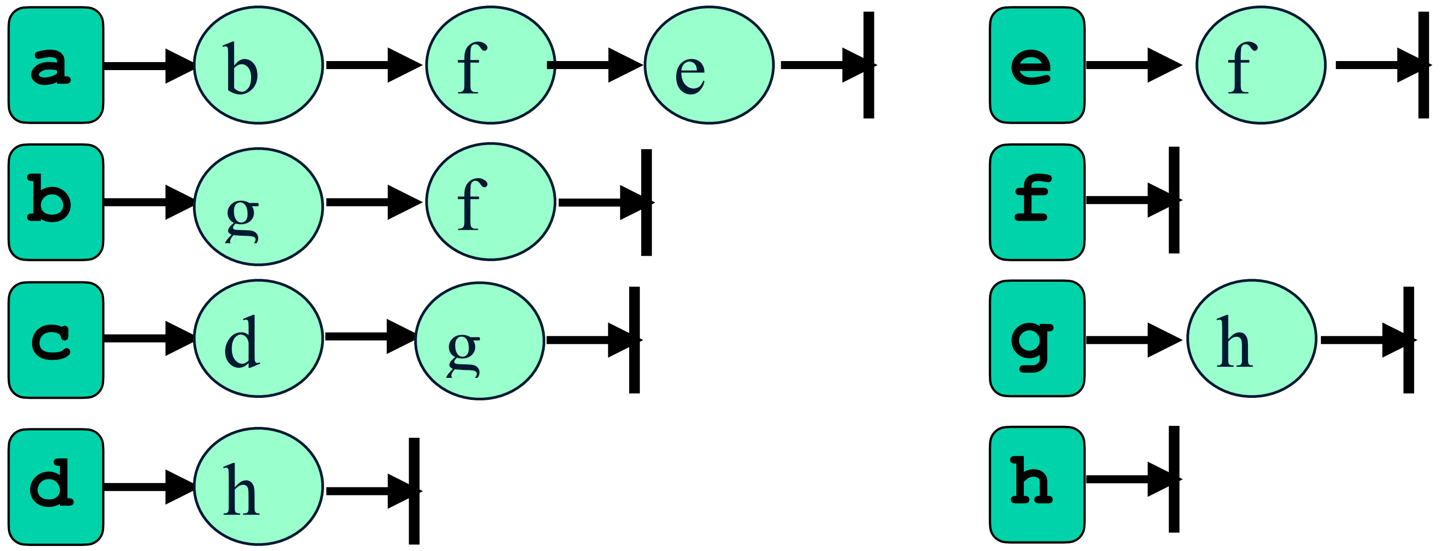
Queue with indegree 0



Topo sort (D&C)-Implementation



Adjacency List



Indegrees

a:0

bb01

c:0

dd01

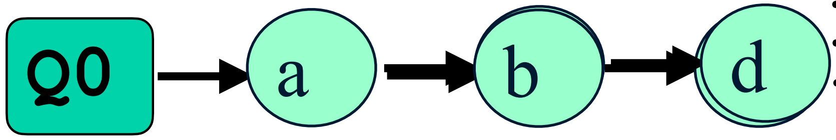
ee01

ff01f:2f:3

gg01g:2

h h0:2:1

Queue with indegree 0



Topo sort (D&C)-Complexity

- Scanning a list of edges to build
  - Adjacency list: `O(|E|)`
  - Indegree list: `O(|E|)`
  - Queue of zero Indegree: `O(|V|)`
- With each iteration of removing front of Q
  - Indegree list is changed
  - node is added to end of Queue of zero indegree
- All the work done in all iterations
  - `O(|E|)` for changing indegree
  - `O(|V|)` for updating Queue of zero indegree
- Total time complexity: `O(|V|+|E|)`

Summary

- Topological order
- Directed acyclic graph
- Directed cyclic graph
- Topo sort using DFS
  - Time complexity: `O(|V|+|E|)`
- Topo sort using node removal
  - Time complexity: `O(|V|+|E|)`