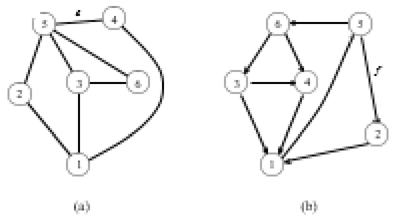
Graph Algorithms

Graphs

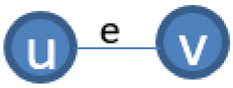


(a) An undirected graph (b) a directed graph.

- An Abstract way of representing connectivity using **nodes**(also called **vertices**) and **edges**
- m edges connect some pairs of nodes
  - Edges can be either directed or undirected
- Nodes and edges can have some auxiliary(보조의) information

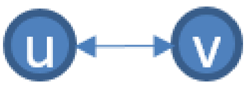
Definitions

- Undirected Graph G



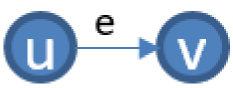
$$V = \{u, v\}, E = \{(u, v)\}$$

  - A pair (V, E), where V is a a finite set of points called vertices and E is a finite set of edges
  - can be thought of as a directed graph



$$V = \{u, v\}, E = \{(u, v), (v, u)\}$$

- Directed Graph



$$V = \{u, v\}, E = \{(u, v)\}$$

  - The edge e is an **ordered** pair (u, v)
  - An edge (u, v) is **incident from** vetex u and is **incident to** vertex v
- A **path** from a vertex v to a vertex u is a sequence $(v_0, v_1, v_2, \ldots, v_k)$ of vertices where $v_0 = v, v_k = u$ and $(v_i, v_{i+1}) \in E$ for $i = 0, 1, \ldots, k-1$
- A vertex u' is **reachable** from vertex u if there is a path p from u to u' in G
- The **length of path** is defined as the number of edges in the path
- A **cycle** is a path where $v_0 = v_k$
- An undirected graph is **connected** if every pair of vertices is connected by a path
- A **forest** is an acyclic(cycle이 없는) graph(tree가 여러개)
, and a **tree** is a connected acyclic graph
- A graph that has weights associated with each edge is called a **weighted graph**

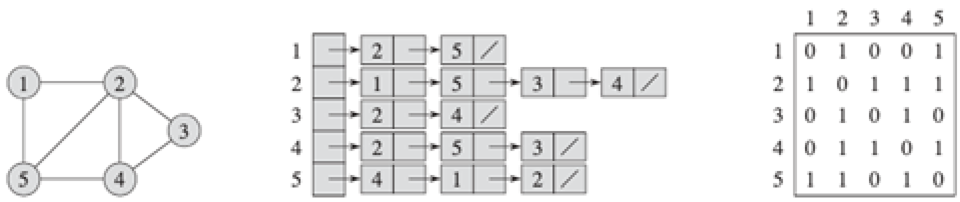
Tree

- A connected acyclic graph
- Most important type of special graphs
  - many problems are easier to solve on trees
- Alternate equivalent definitions
  - A connected graph with $n > 1$ edges (where n is a number of vertices)
  - An acyclic graph with $n > 1$ edges
  - There is exactly **one path** between every pair of nodes
  - An acyclic graph but adding any edge results in a cycle
  - A connected graph but removing any edge disconnects it

Representation of a graph

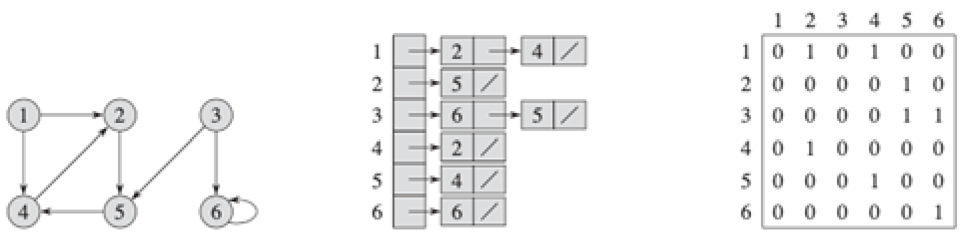
- Graphs can be represented by their adjacency matrix or adjacency list
- Adjacency matrices have a value $a_{ij} = 1$ if nodes i and j share an edge; 0 otherwise.
  In case of a weighted graph, $a_{ij} = w_{ij}$, the weight of the edge
- The adjacency list representation of a graph $G = (V, E)$ consists of an array $Adj[1...|V|]$ of lists. Each list $Adj[v]$ is a list of all vertices adjacent to v
- For a graph with n nodes, adjacency matrices take $\theta(n^2)$ space and adjacency list takes $\theta(|E| + |V|)$ space
- 교과서에서는 대부분 **Adjacency List** 표현을 가정

Undirected Graph



(a) (b) (c)

Directed Graph

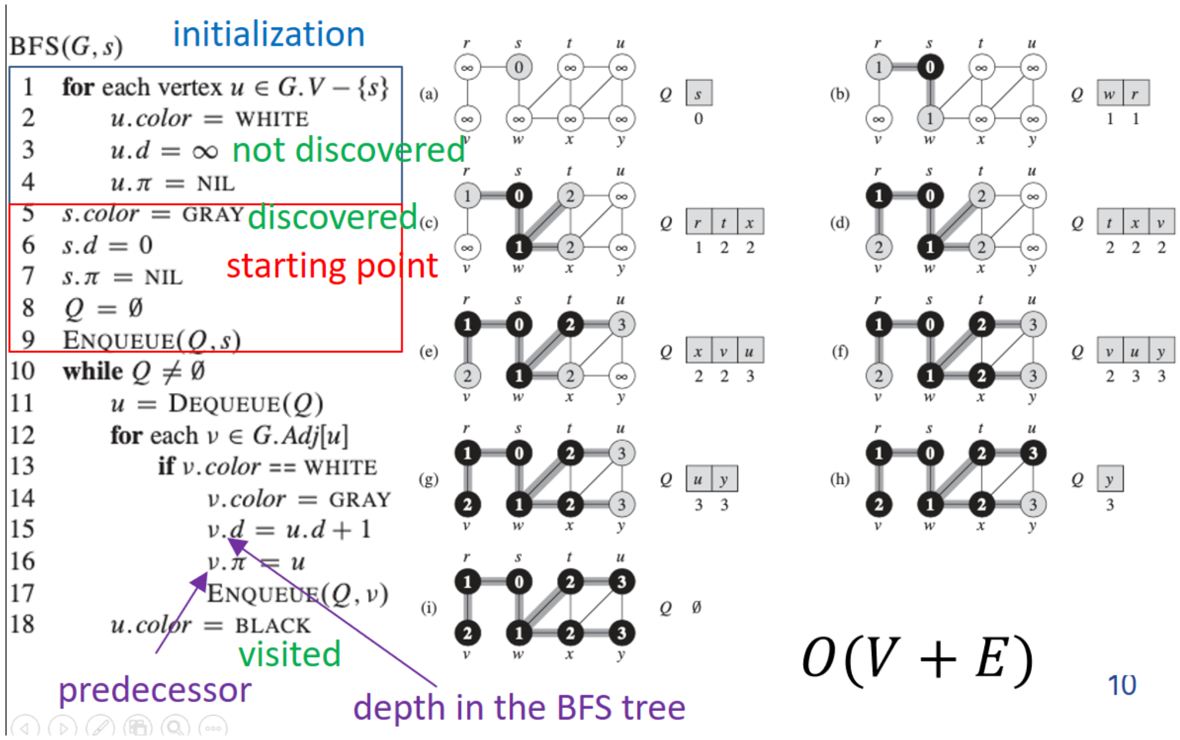


(a) (b) (c)

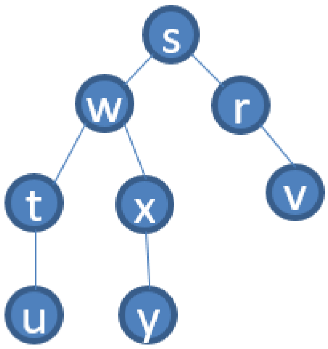
Graph Traversal

- The most basic graph algorithm that visit nodes of a graph in certain ored
- Used as a subroutine in many other algorithms
  - 특별한 언급이 없으면 vertex는 **알파벳 순**으로 처리

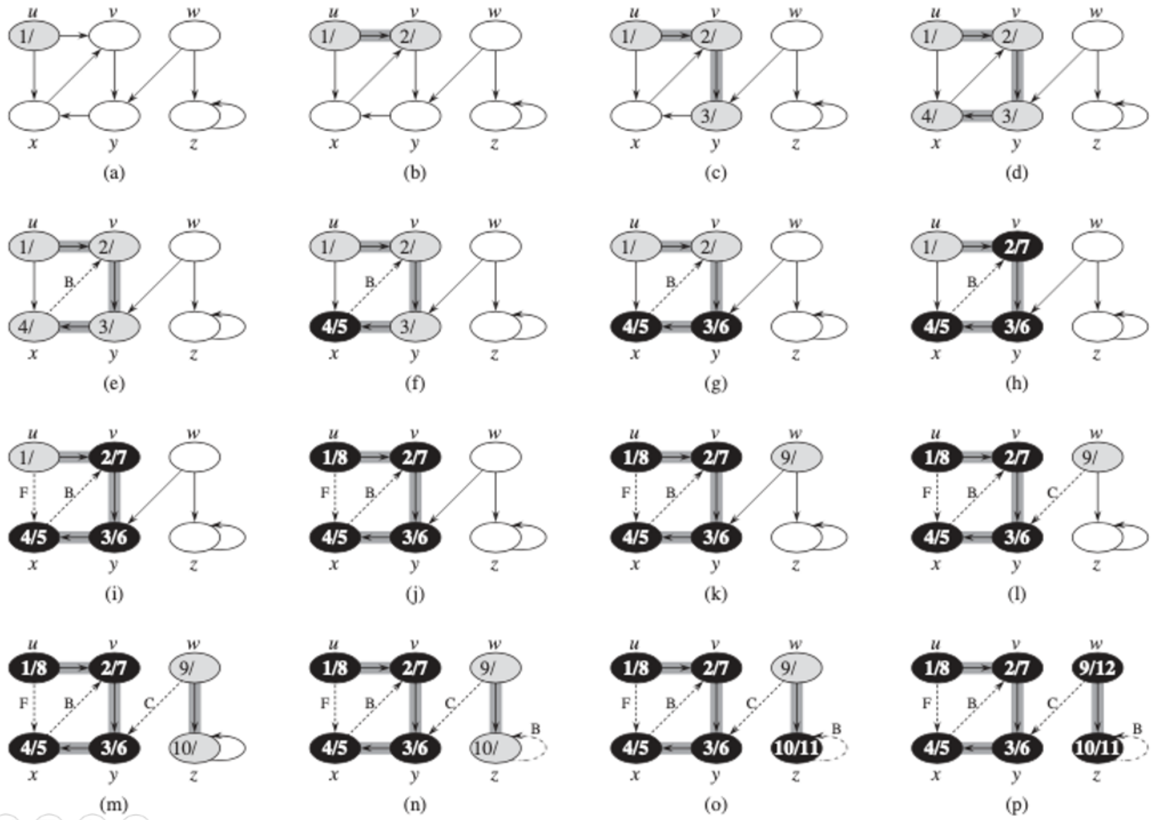
Breadth-First Search(BFS) : uses queue



- vertex s로부터 reachable vertex v에 대한 shortest path distance $\delta(s, v)$를 모두 계산한다
  그 값은 $v.d$이다

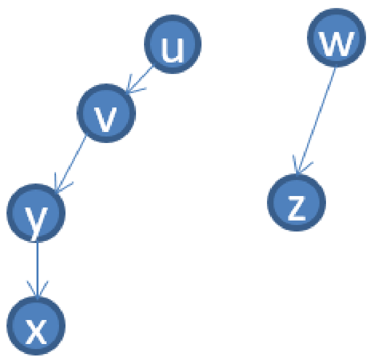


Depth-First Search(DFS) : uses recursion(stack)



- 앞은 Starting Time, 뒤는 Finish Time(back tracking)

DFS Forest



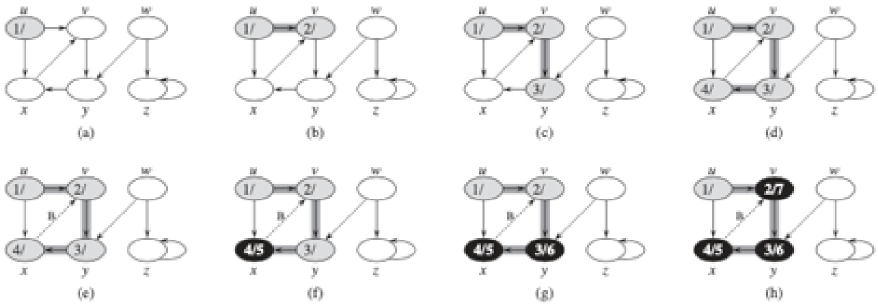
DFS Algorithm

DFS algorithm

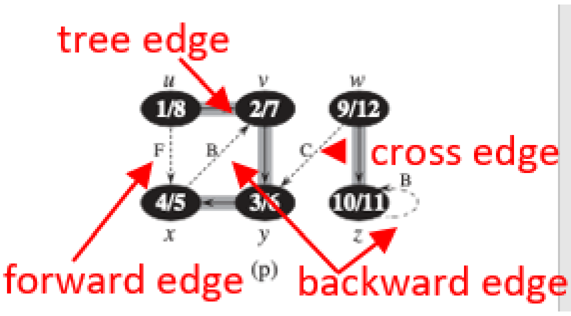
```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL        not discovered
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

$O(V+E)$

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY        discovered
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK       finished(=visited)
time = time + 1
u.f = time
```



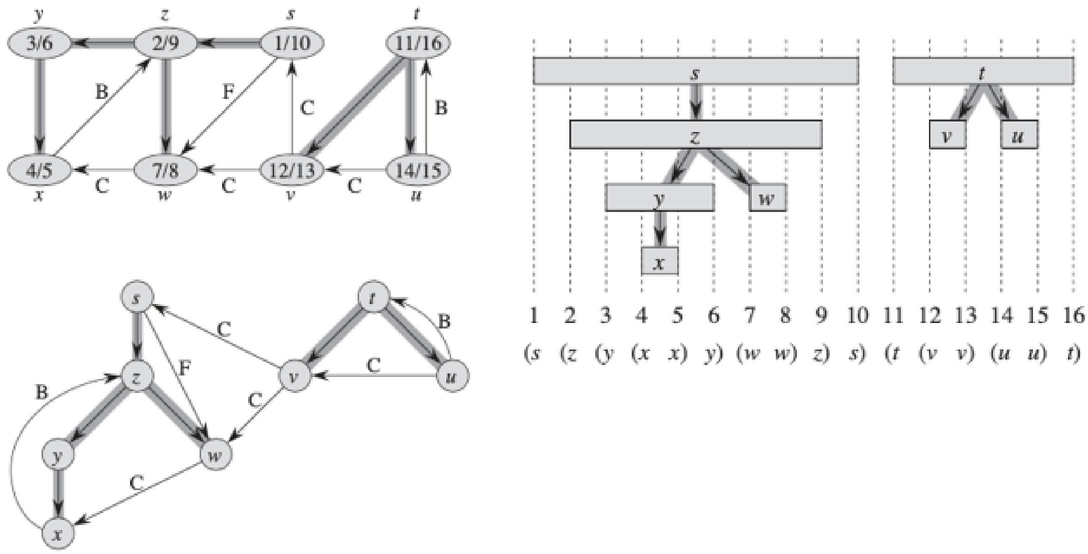
14



- Backward edge : 돌아가는 edge
- Cross edge : 서로 다른 트리에서 access 하는 경우
- Forward edge : 트리 내에서 선택되지 않은 edge?
- Tree edge : 트리에 포함되는 edge

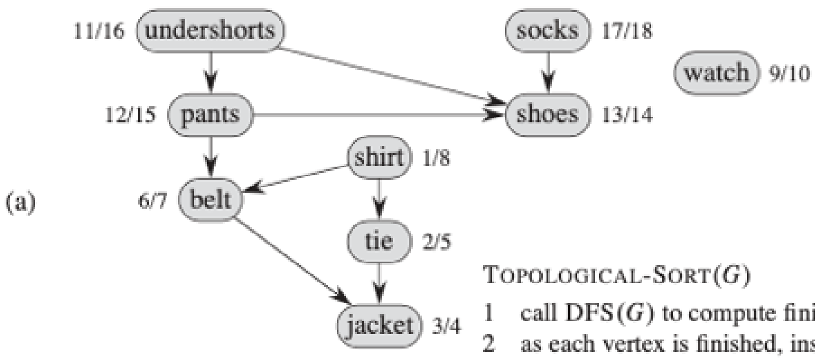
Properties

- 다음과 같은 timestamped directed graph가 만들어진다



Topological Sort

- DAG : Directed Acyclic Graph(사이클이 없는 Directed Graph)
  - Total Order가 있는 집합
    - Sorting 그냥 하면 됨
  - **Partial Order가 있는 집합**(위의 예)
    - 대소관계가 있는 것들끼리 관계를 그래프로 표현




```
TOPOLOGICAL-SORT(G)
1  call DFS(G) to compute finishing times v.f for each vertex v
2  as each vertex is finished, insert it onto the front of a linked list
3  return the linked list of vertices
```

- Finish Time에 따라서 리스트의 앞에 집어넣는다

Theorem 22.12

- TOPOLOGICAL-SORT algorithm produces a topological sort of the directed acyclic graph provided as its input

```
TOPOLOGICAL-SORT(G)
1  call DFS(G) to compute finishing times v.f for each vertex v
2  as each vertex is finished, insert it onto the front of a linked list
3  return the linked list of vertices
```

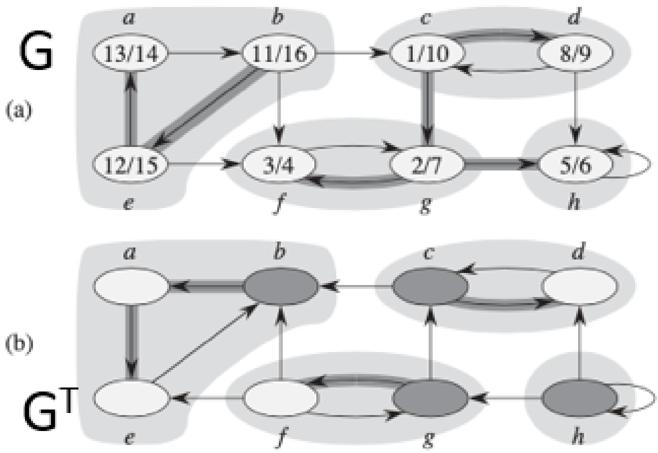
Proof Suppose that DFS is run on a given dag G = {V,E} to determine finishing times for its vertices. It suffices to show that for any pair of distinct vertices u,v in V , if G contains an edge from u to v, then v.*f* < u.*f*.

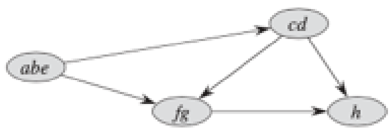
Strongly connected components

- Undirected graph
  - **connected** if every vertex is reachable from all other vertices
  - **connected components** of a graph are the equivalence classes of vertices under the 'is reachable from' relation
- Directed graph
  - **strongly connected** if every two vertices are reachable from each other
  - **strongly connected components** of a directed graph are the equivalence classes of vertices under the 'are mutually reachable' relation

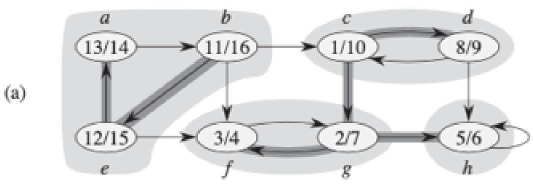
STRONGLY-CONNECTED-COMPONENTS(G)

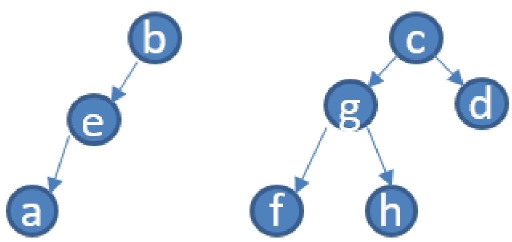
1 call DFS(G) to compute finishing times $u.f$ for each vertex u
2 compute G^{T}
3 call DFS(G^{T}), but in the main loop of DFS, consider the vertices
    in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a
    separate strongly connected component





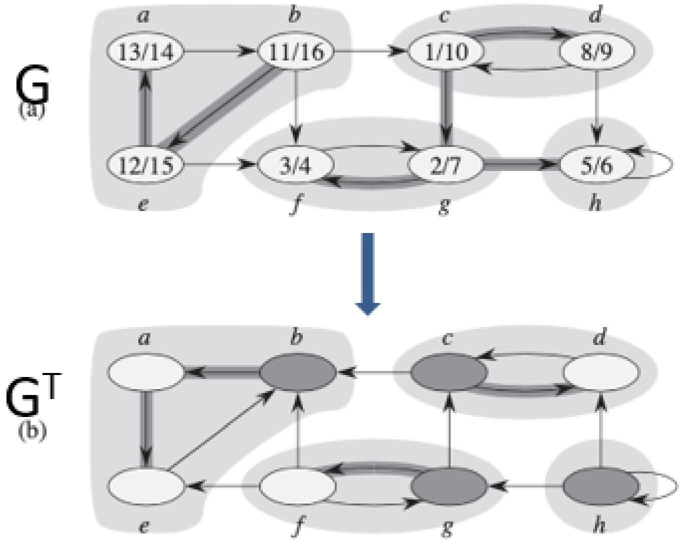
DFS(G)





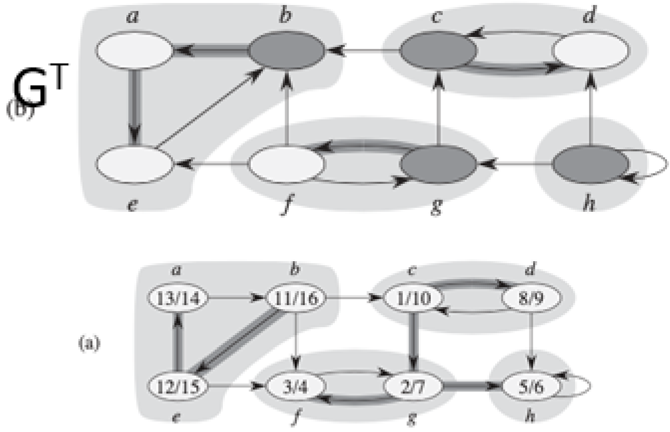
Compute G^T

- All directions are reveresed

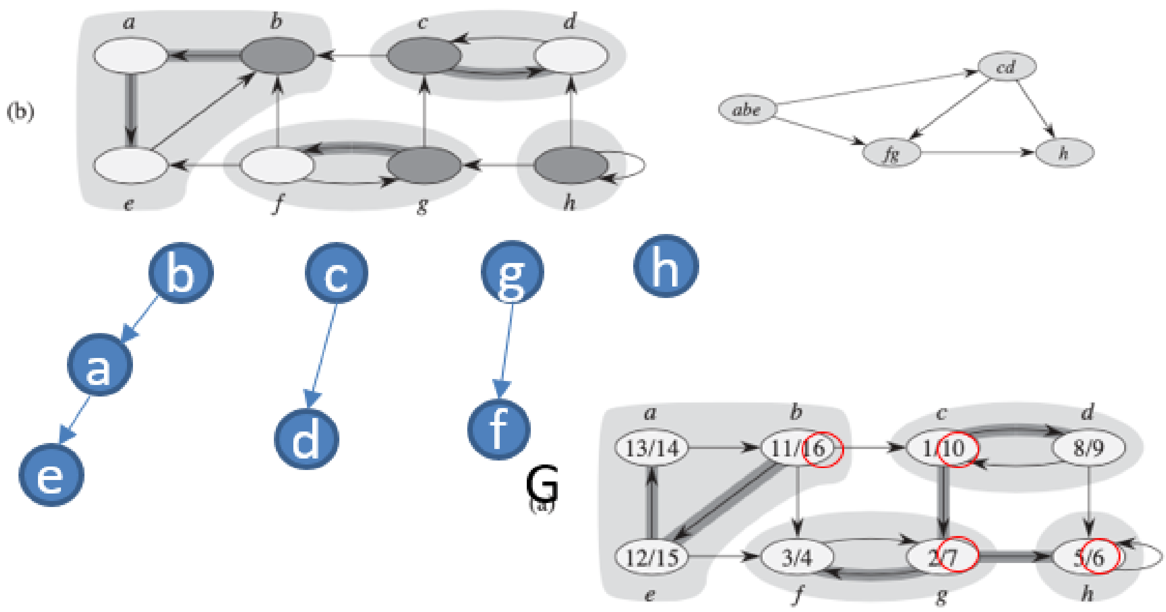


DFS(G^T)

- Vertices are selected in order of decreasing u.f



SCC



Euler Tour

- An Euler tour of a strongly connected, directed graph $G = (V, E)$ is a cycle that traverses each edge of G exactly once, although it may visit a vertex more than once
    - G has an Euler tour iff in-degree(v) = out-degree(v) for all $v \in V$
    - The Euler tour algorithm runs in $O(E)$