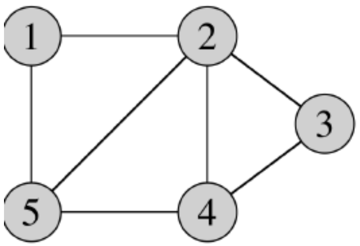
Spring 2012

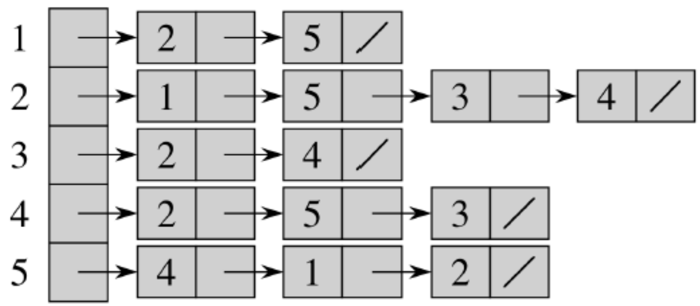
ICS621 Graph Algorithms

Lipyeow Lim

Data Structures for Graphs



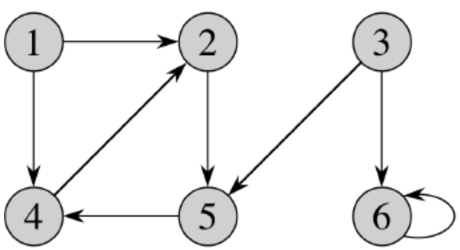
(a)



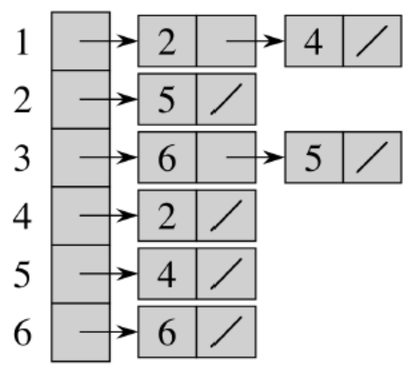
(b)

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)



(a)



(b)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

BFS for Graphs

BFS(V, E, s)

for each $u \in V - \{s\}$

$\quad u.d = \infty$

$s.d = 0$

$Q = \emptyset$

ENQUEUE(Q, s)

while $Q \neq \emptyset$

$\quad u = $ DEQUEUE(Q)

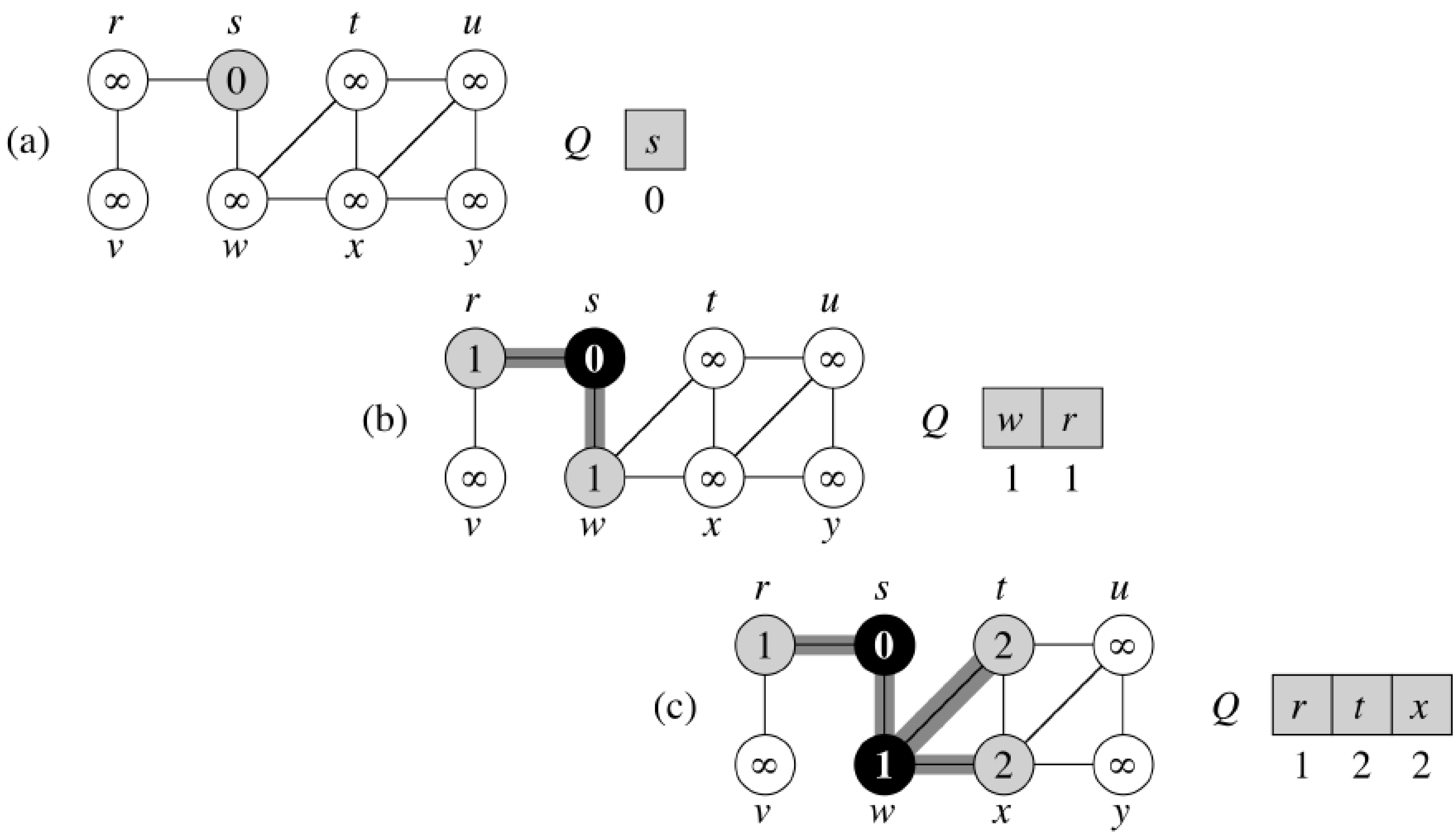
$\quad$ **for** each $v \in G.Adj[u]$

$\quad\quad$ **if** $v.d == \infty$

$\quad\quad\quad v.d = u.d + 1$

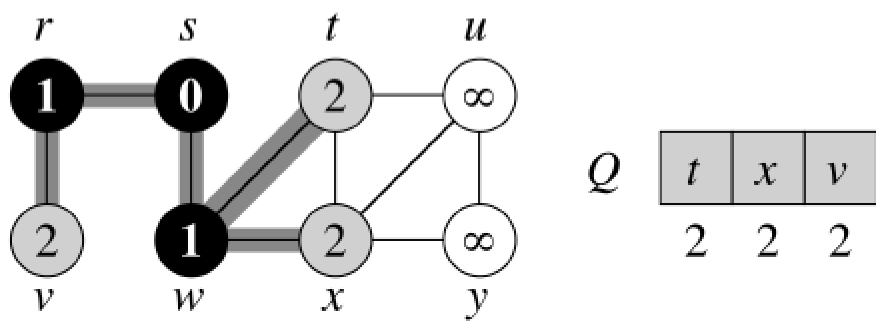
$\quad\quad\quad$ ENQUEUE(Q, v)

BFS Trace 1/3

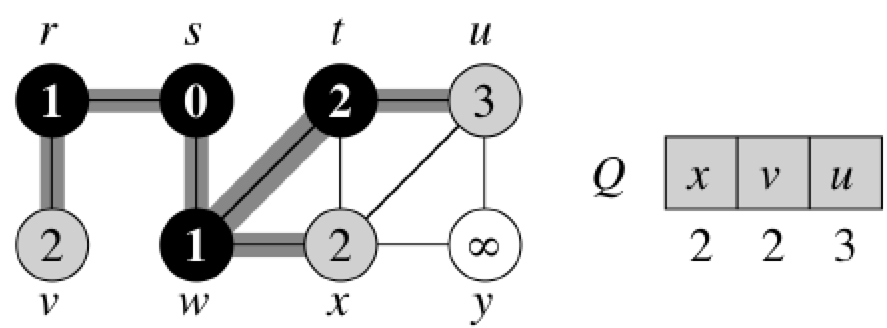


BFS Trace 2/3

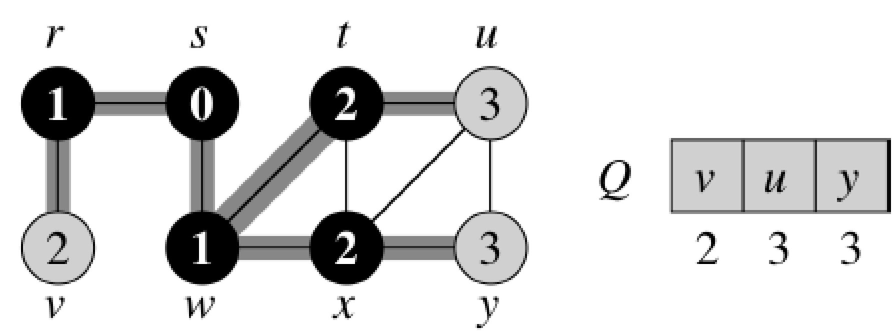
(d)



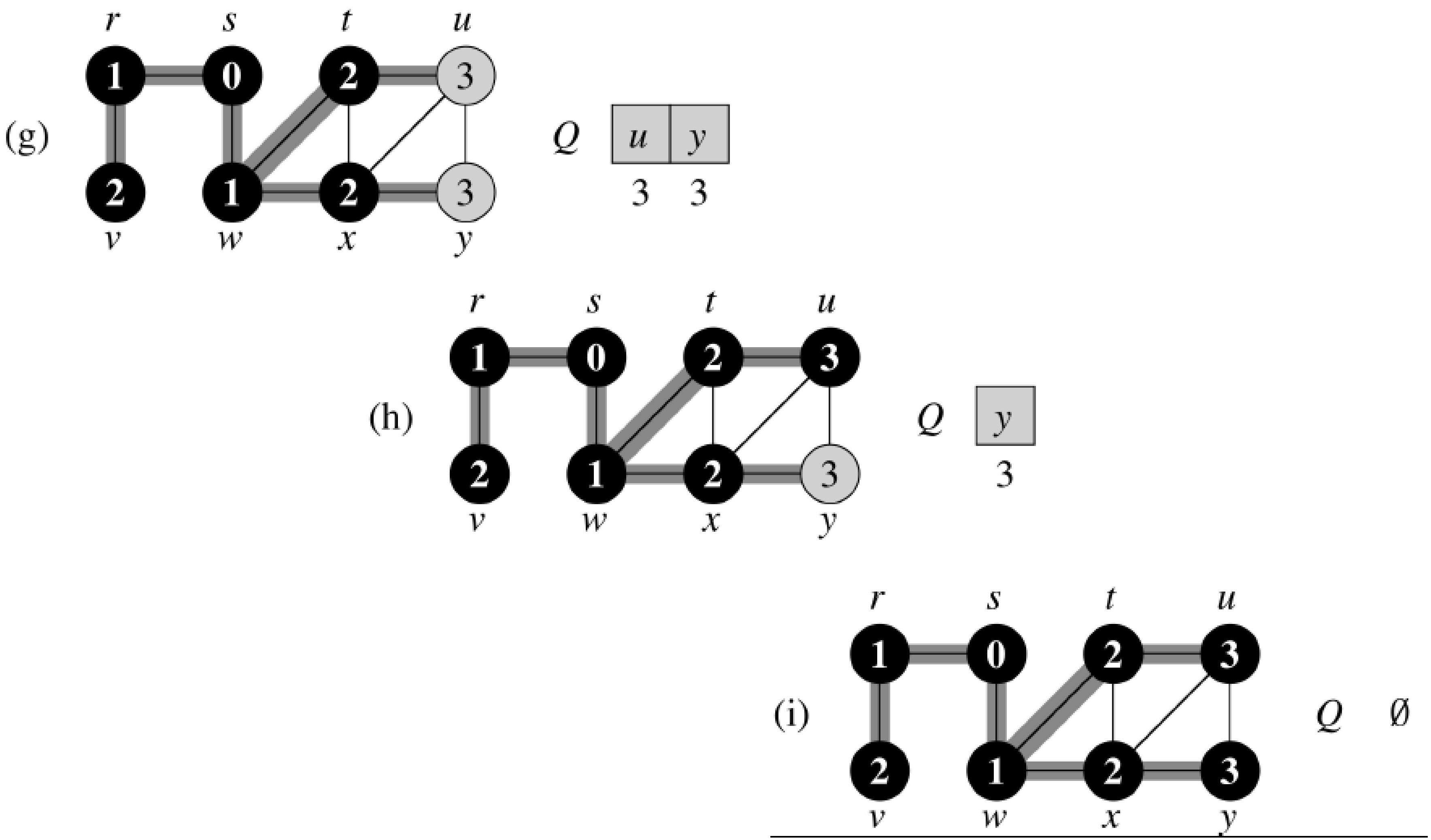
(e)



(f)



BFS Trace 3/3



(g)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q u y
3 3
(h)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q y
3
(i)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q Ø

DFS for Graphs

DFS(G)

for each $u \in G.V$

$\quad u.color = \text{WHITE}$

$time = 0$

for each $u \in G.V$

$\quad$ **if** $u.color == \text{WHITE}$

$\quad\quad$ DFS-VISIT(G, u)

DFS-VISIT(G, u)

$time = time + 1$

$u.d = time$

$u.color = \text{GRAY}$

for each $v \in G.Adj[u]$

$\quad$ **if** $v.color == \text{WHITE}$

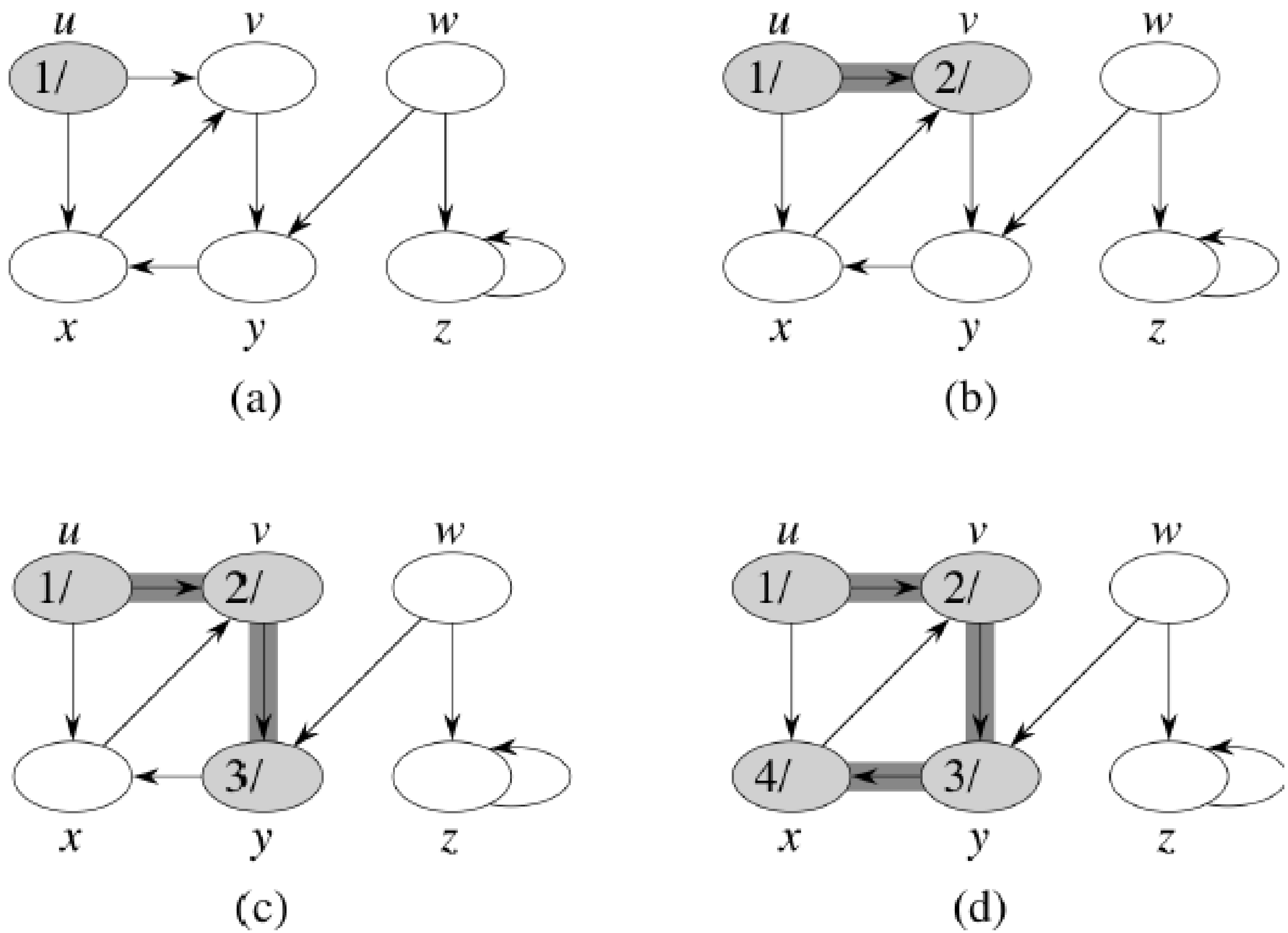
$\quad\quad$ DFS-VISIT(v)

$u.color = \text{BLACK}$

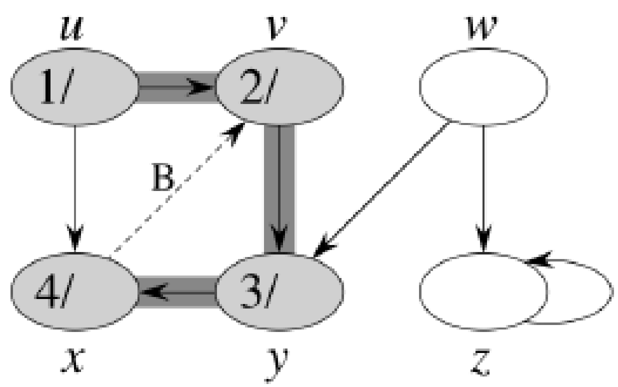
$time = time + 1$

$u.f = time$

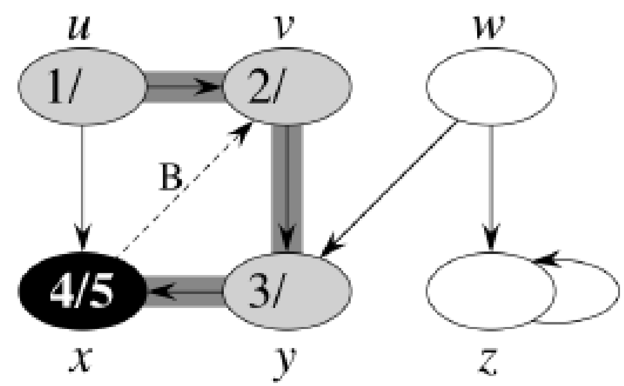
DFS Trace 1/4



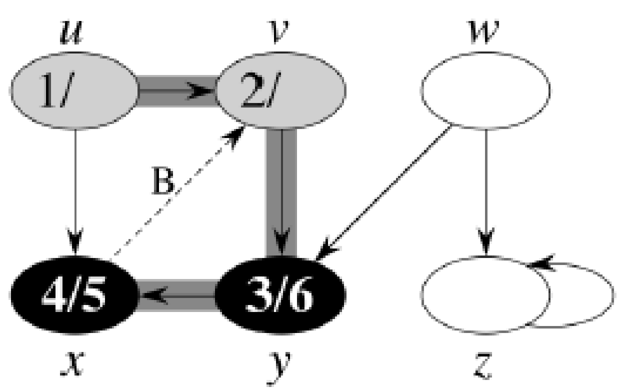
DFS Trace 2/4



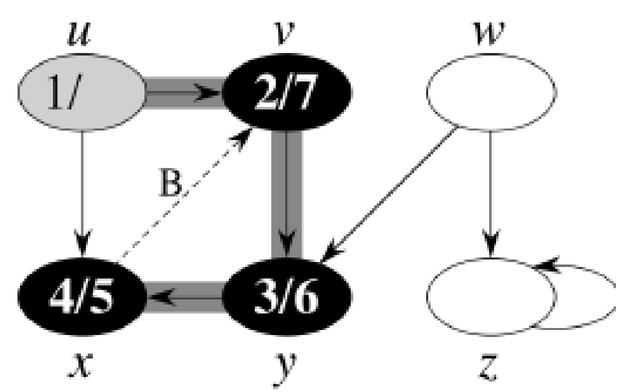
(e)



(f)

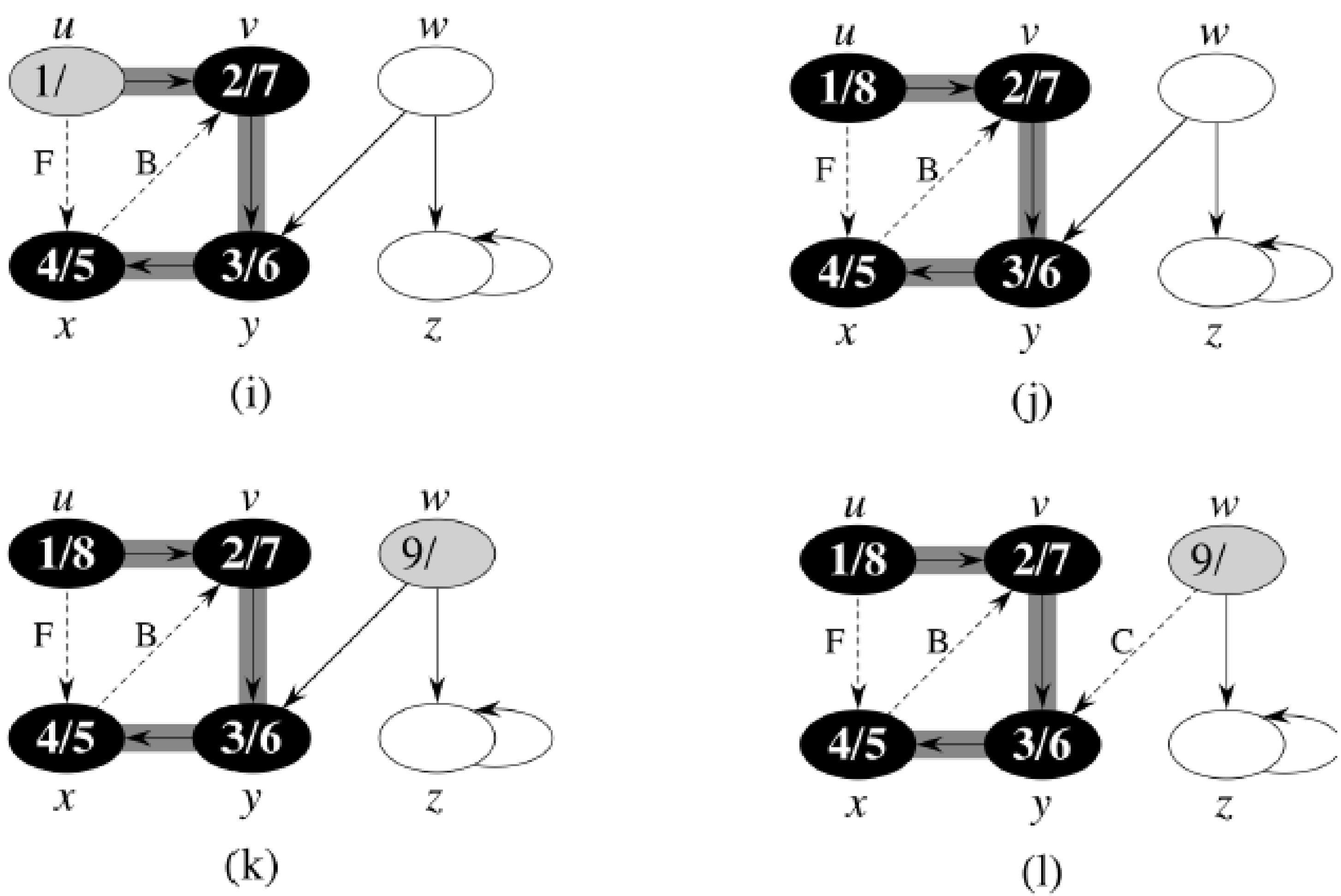


(g)

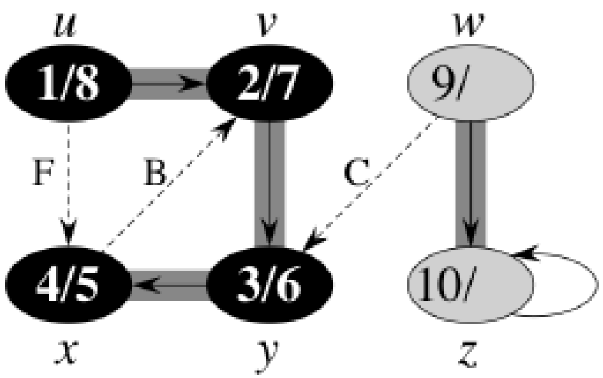


(h)

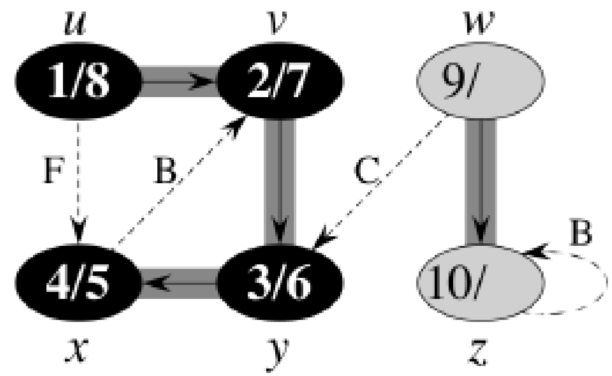
DFS Trace 3/4


u
v
w
1/
2/7
F
B
4/5
3/6
x
y
z
(i)
u
v
w
1/8
2/7
F
B
4/5
3/6
x
y
z
(j)
u
v
w
1/8
2/7
9/
F
B
4/5
3/6
x
y
z
(k)
u
v
w
1/8
2/7
9/
F
B
C
4/5
3/6
x
y
z
(l)

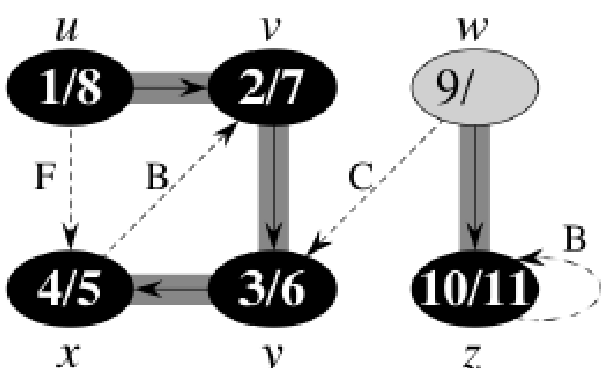
DFS Trace 4/4



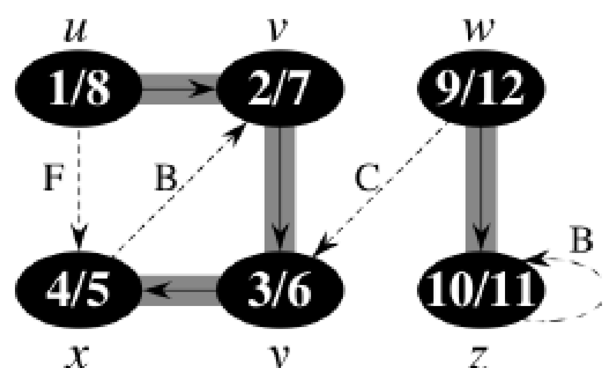
(m)



(n)

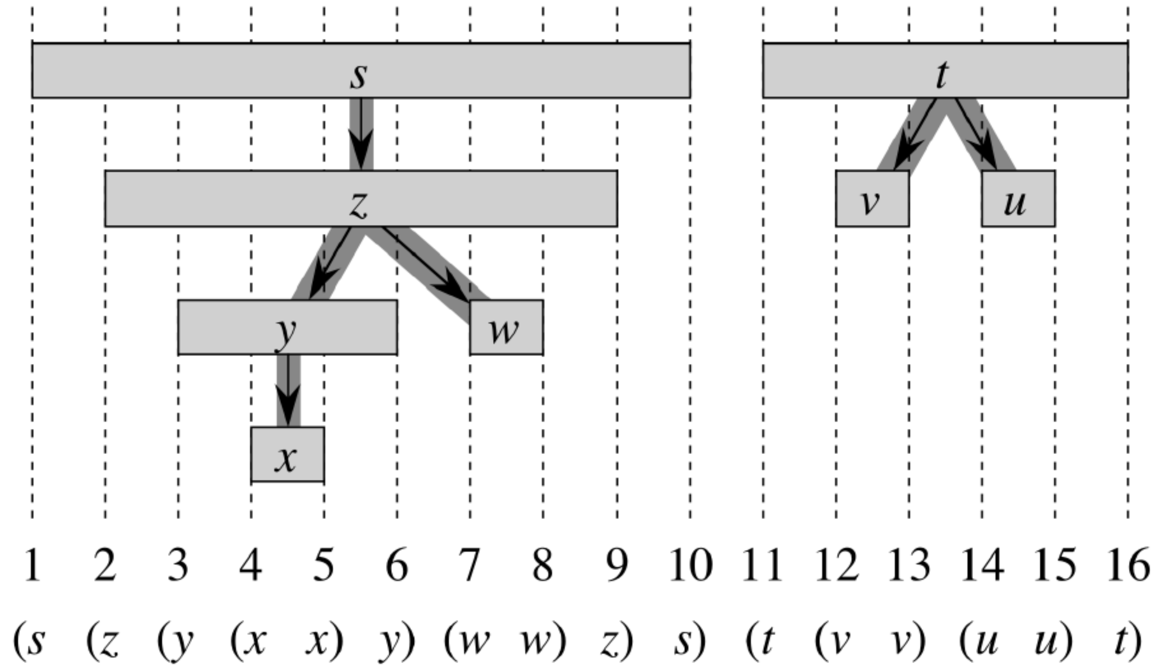


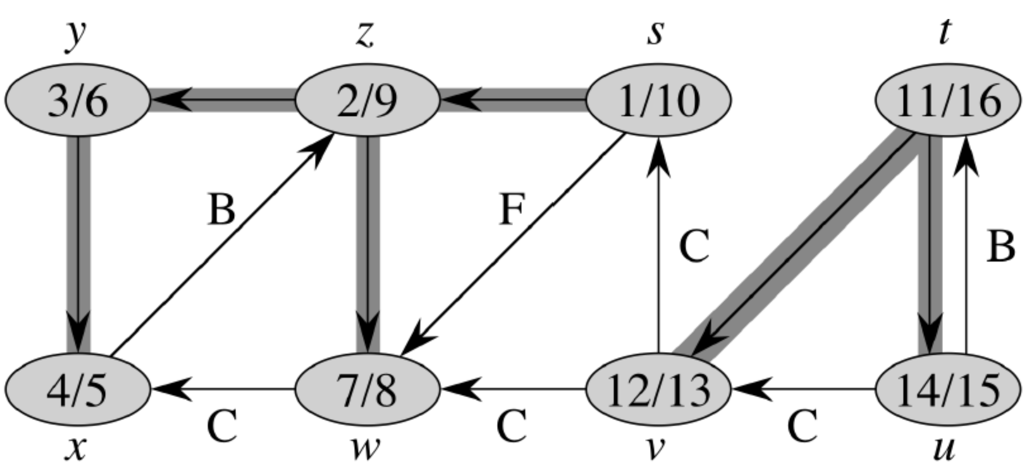
(o)

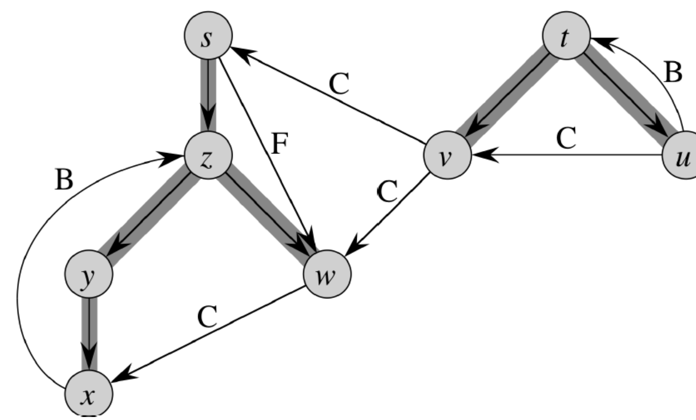


(p)

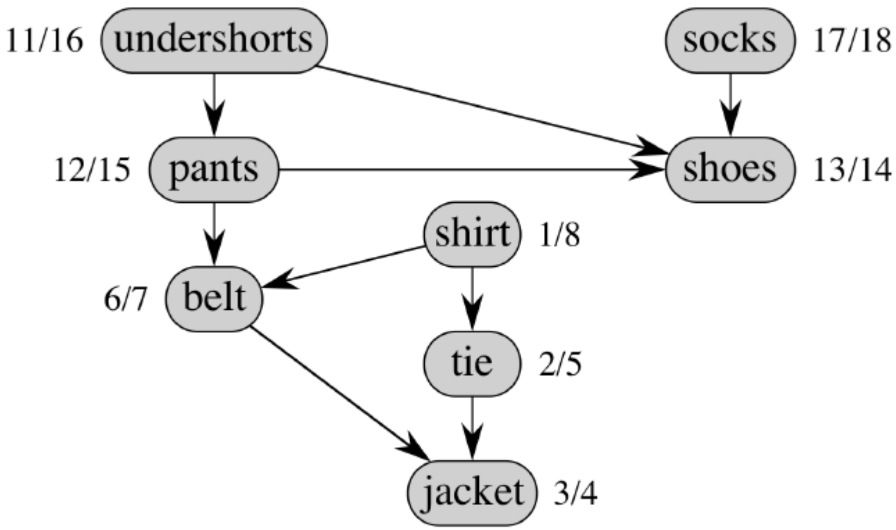
DFS Timestamps







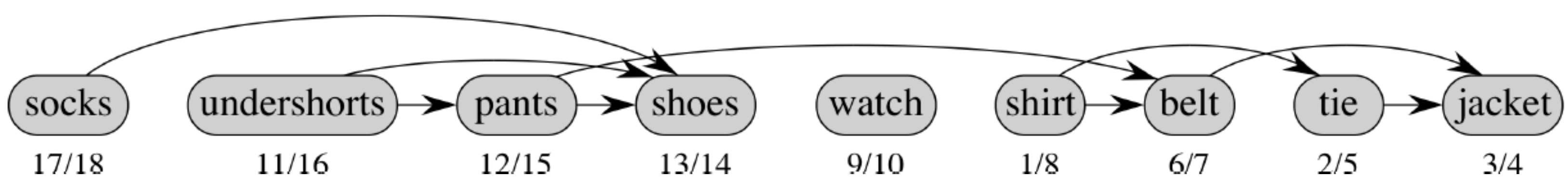
Topological Sort



- **Input**: directed acyclic graph G=(V,E)
- **Output**: a linear ordering of the vertices s.t. if (u,v) in E, then u precedes v

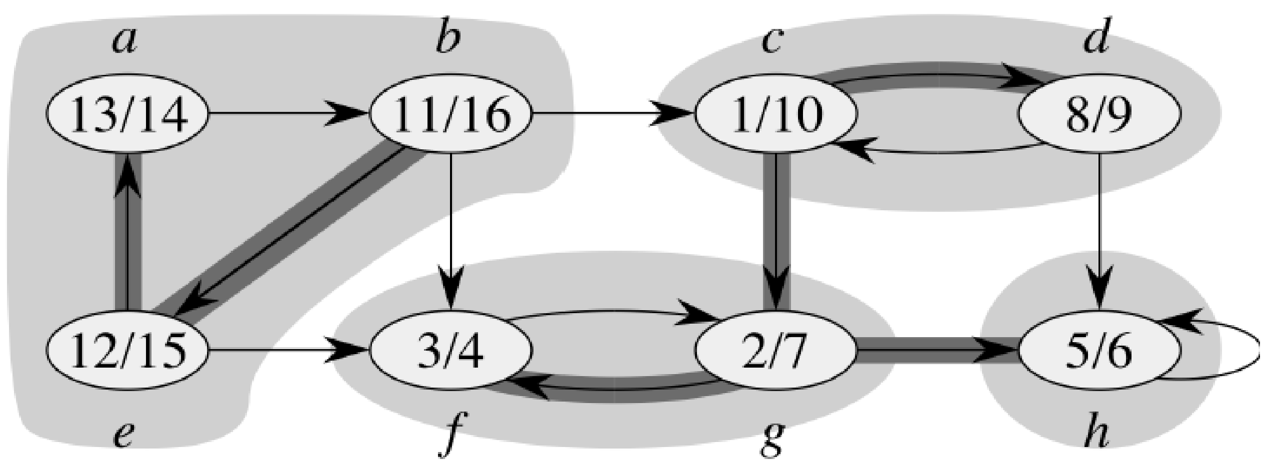
TOPOLOGICAL-SORT(G)

call DFS(G) to compute finishing times $v.f$ for all $v \in G.V$
output vertices in order of *decreasing* finishing times

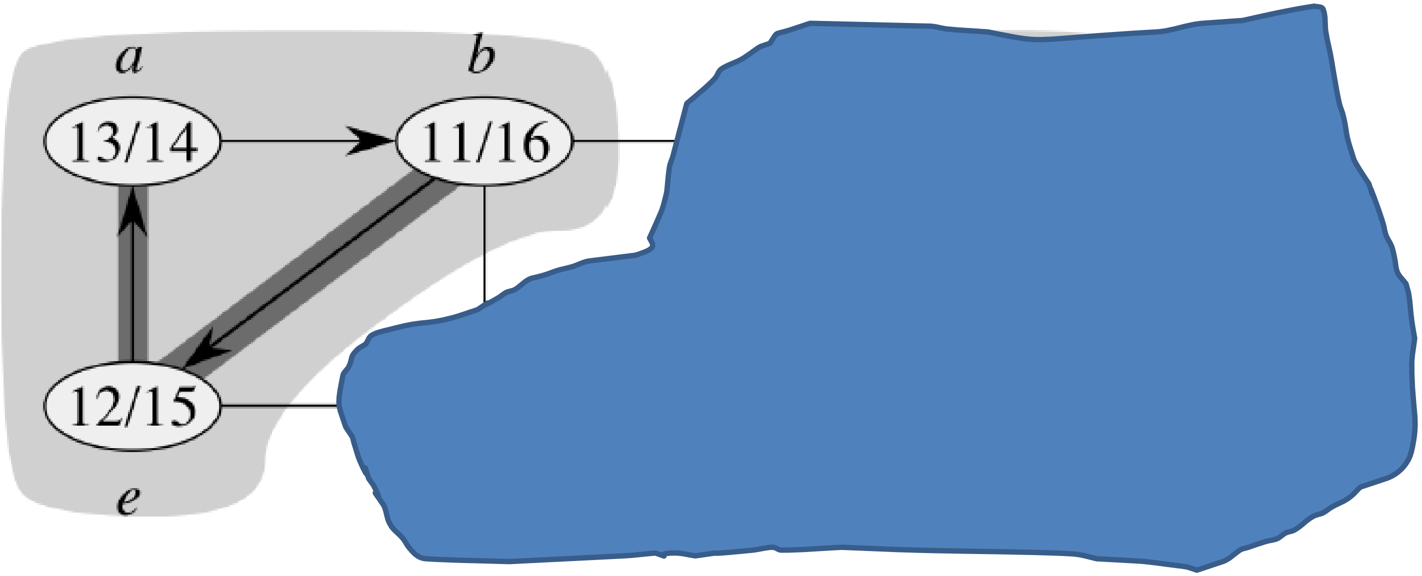


Strongly Connected Components

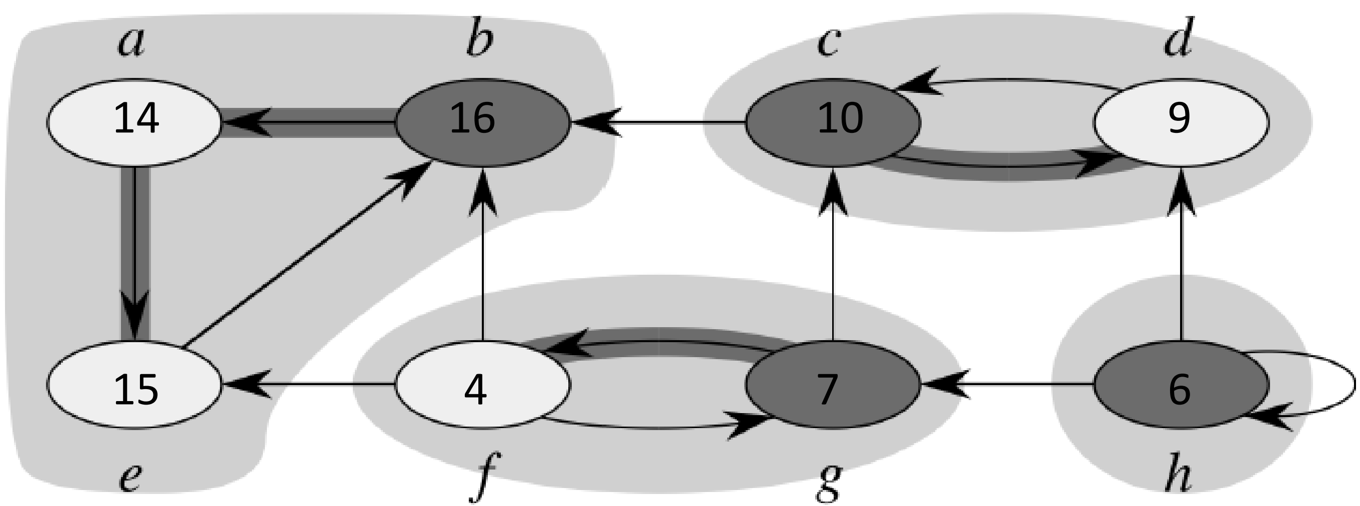
- **Input**: Directed Graph G=(V,E)
- **Output**: A collection of all *strongly connected components* (SCC) of G
- A SCC of G is a maximal set of vertices C subset of V s.t. for all pair (u,v) in C, there is a path from u to v and from v to u.



Finding Strongly Connected Components



Edges Reversed

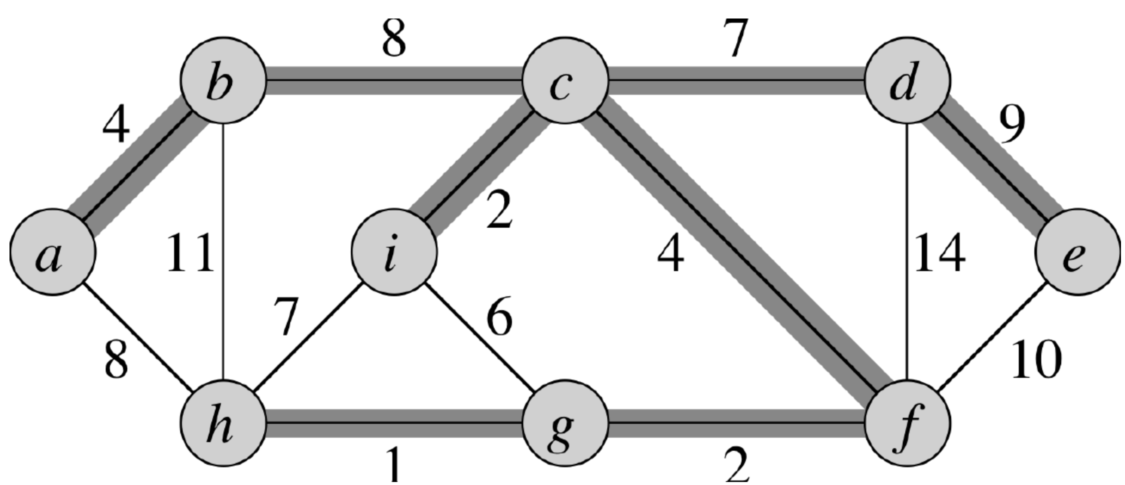


Algorithm for finding SCC

1. DFS(G) to compute finishing times for each vertex
2. Compute G’ transpose of G (reverse edges)
3. DFS(G’) using decreasing order of finishing times
4. Output vertices of each tree in DFS forest formed in DFS(G’)

Minimum Spanning Trees

- **Input**: undirected graph G=(V,E) with edge weights w(u,v)
- **Output**: a tree T=(V’, E’) s.t.
  - T is a tree
  - V’ = V (hence spanning V)
  - Sum of w(u,v) for all (u,v) in E’ is minimal



Kruskal's Algo

KRUSKAL(G, w)

$A = \emptyset$
for each vertex $v \in G.V$
    MAKE-SET(v)
sort the edges of $G.E$ into nondecreasing order by weight w
for each (u, v) taken from the sorted list
    if FIND-SET$(u) \neq$ FIND-SET(v)
        $A = A \cup \{(u, v)\}$
        UNION(u, v)
return A

Keep adding edges
- with smallest weight
- does not form cycles

Until V-1 edges are added.

- Disjoint Set Ops = O(α(V)) (Ch21)
- E = O(V*V)
- O(E lg V)

Prim's Algo

$\text{PRIM}(G, w, r)$

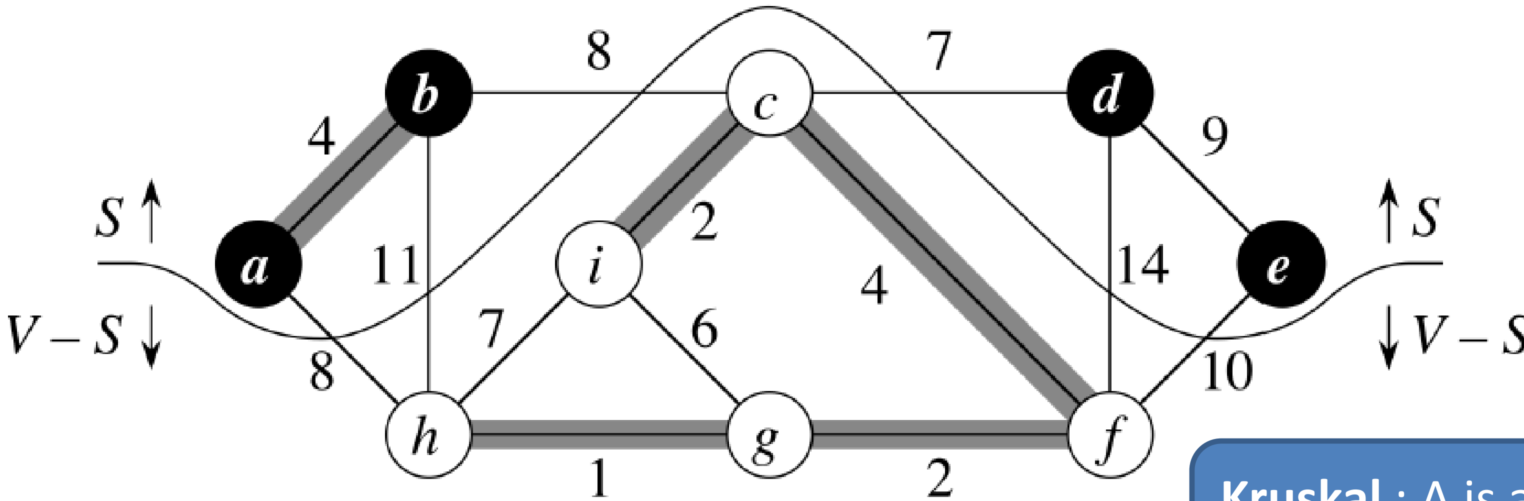
$Q = \emptyset$
for each $u \in G.V$
  $u.key = \infty$
  $u.\pi = \text{NIL}$
  $\text{INSERT}(Q, u)$
$\text{DECREASE-KEY}(Q, r, 0)$ // $r.key = 0$
while $Q \neq \emptyset$
  $u = \text{EXTRACT-MIN}(Q)$
  for each $v \in G.Adj[u]$
    if $v \in Q$ and $w(u, v) < v.key$
      $v.\pi = u$
      $\text{DECREASE-KEY}(Q, v, w(u, v))$

Grow a tree by adding edges
- with smallest weight
- does not form cycles

Fib Heap
- Dec Key = O(1)
- Ext Min = O(lg V)

=> O(E + V lg V)

Generic Framework



GENERIC-MST(G, w)

$A = \emptyset$
while A is not a spanning tree
  find an edge (u, v) that is safe for A
  $A = A \cup \{(u, v)\}$
return A

Kruskal : A is a forest
Prim : A is a tree

Kruskal : safe edge connects 2 components
Prim : safe edge connects A to a new vertex