Graph Algorithms (Chapter 22)

University of Stavanger

Graph Algorithms (Chapter 22)

Goals

Graph definitions

Graph representations

Search on graph

Topological sort

Strongly connected components

Graph Algorithms (Chapter 22.1)

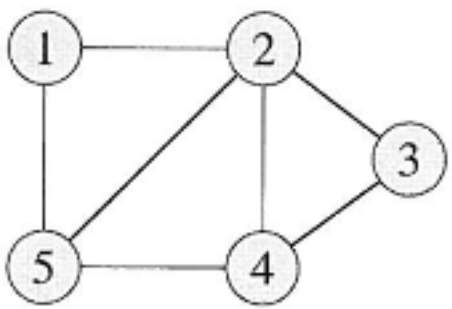
Definitions

- $G = (V, E)$, where V is the set of vertices, we may call them nodes sometimes, and E the set of edges.
- In general, edge $(u, v) \in E$ can be directed or undirected, and can also have a weight $w(u, v)$ as the weight value for the edge (u, v), or simply $(u, v) . w$
- If an edge (u, v) has an attribute f we will denote it as $(u, v) . f$
- If a vertex v as an attribute d, we will denote it $v . d$ or $d[v]$

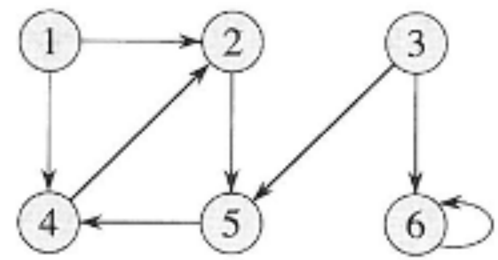
Graph Algorithms (Chapter 22.1)

University of Stavanger

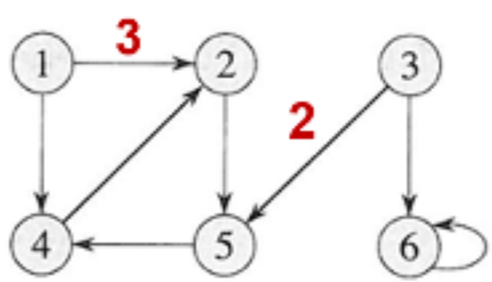
Examples



Undirected graph



Directed graph

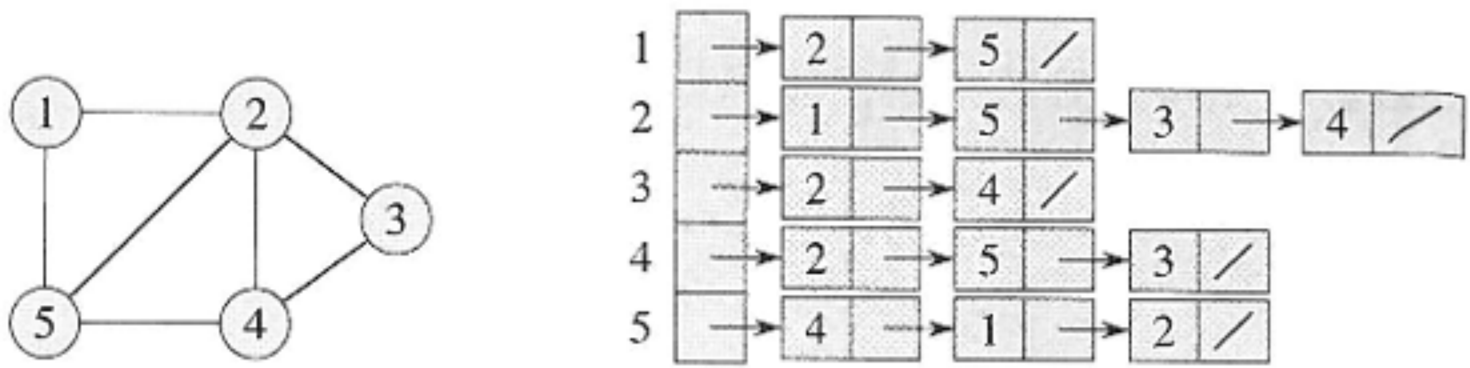


Weighted and directed graph

Graph Algorithms (Chapter 22.1)

University of Stavanger

Graph representation (Adjacency-list)

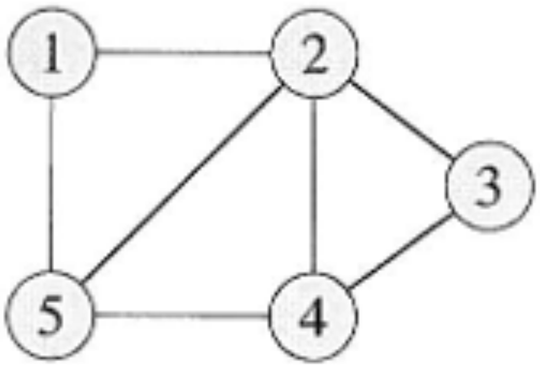


(a) (b)

$\forall u \in V, Adj[u]$ contains all vertices v to which u has an edge $(u, v) \in E$

Graph Algorithms (Chapter 22.1)

Directed graph representation (Adjacency-matrix)



(a)

	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)

$A = (a_{ij})$, $a_{ij} = 1$ if $(i, j) \in E$, $a_{ij} = 0$, otherwise

Graph Algorithms (Chapter 22.1)

Pratical considerations

Given a "connected" directed and undirected graph $G = (V, E)$:

1. What is the maximum number of edges that G can have?

2. What is the minimum number of edges that G can have?

3. What is the space required to represent G?

Graph Algorithms (Chapter 22.2)
Breadth Fist Search (BFS)

A way to explore a graph but:

How does it work?

What is its time complexity?

What other properties it has?

Graph Algorithms (Chapter 22.2)
Breadth Fist Search (BFS)

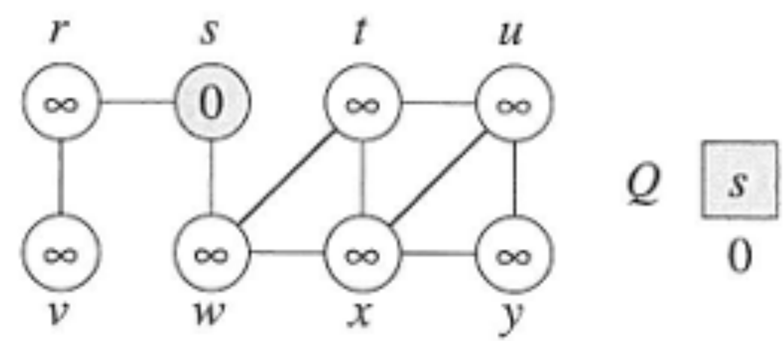
```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

Graph Algorithms (Chapter 22.2)
Breadth Fist Search (BFS)

University of Stavanger

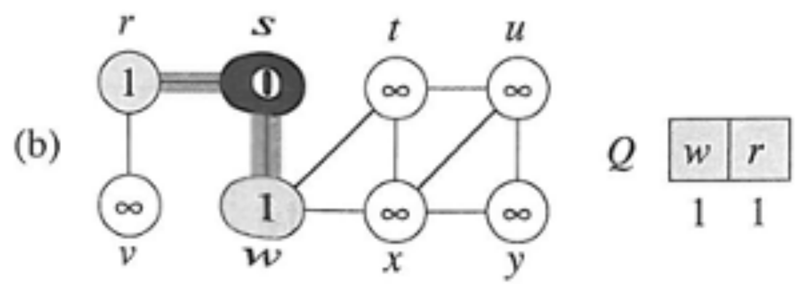
```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

(a)



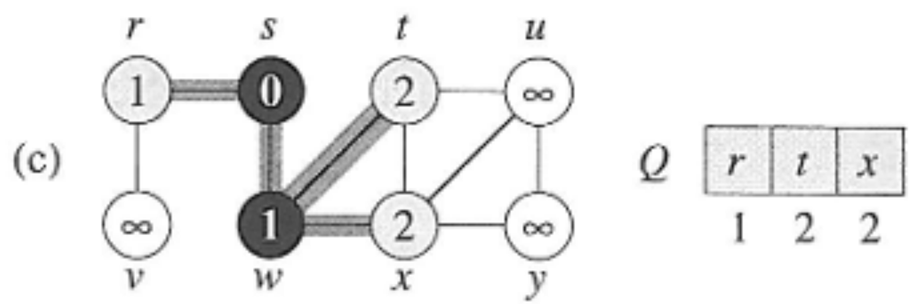
Graph Algorithms (Chapter 22.2) Breadth Fist Search (BFS)

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



Graph Algorithms (Chapter 22.2) Breadth Fist Search (BFS)

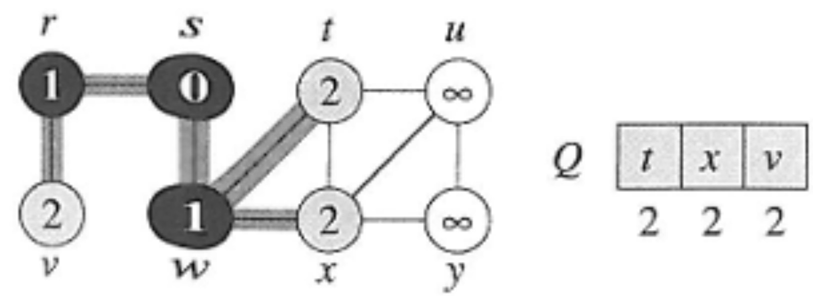
```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



Graph Algorithms (Chapter 22.2)
Breadth Fist Search (BFS)


```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

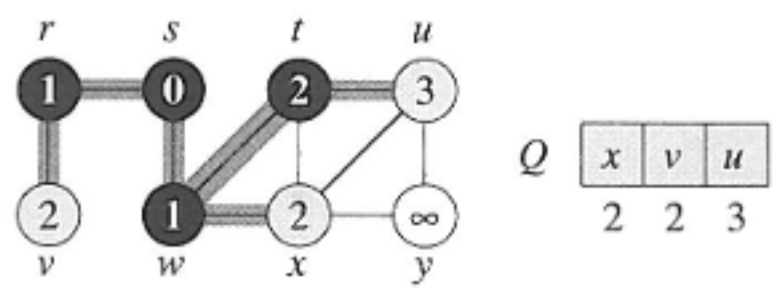
(d)



Graph Algorithms (Chapter 22.2) Breadth Fist Search (BFS)

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

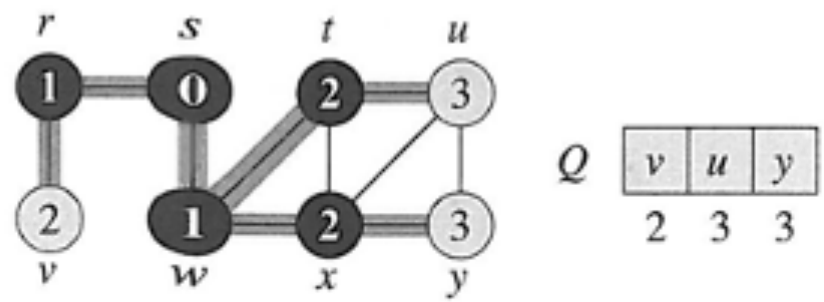
(e)



Graph Algorithms (Chapter 22.2)
Breadth Fist Search (BFS)


```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

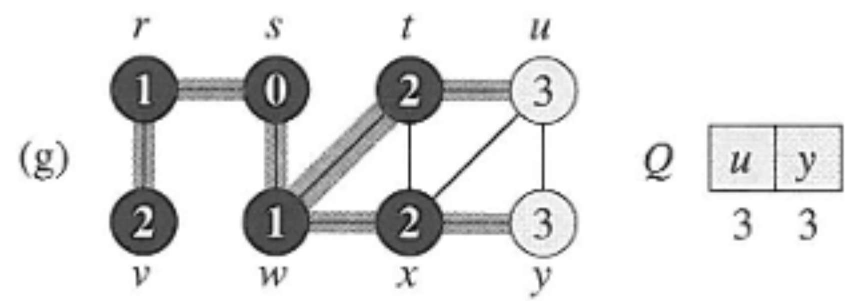
(f)



Graph Algorithms (Chapter 22.2)
Breadth Fist Search (BFS)

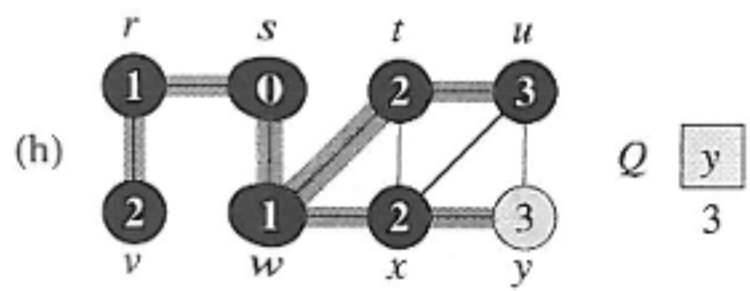
University of Stavanger

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



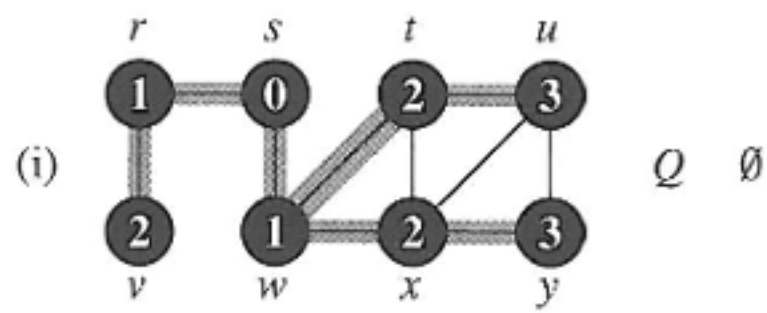
Graph Algorithms (Chapter 22.2) Breadth Fist Search (BFS)

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



Graph Algorithms (Chapter 22.2) Breadth Fist Search (BFS)

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```



Graph Algorithms (Chapter 22.2) Breadth Fist Search (BFS)

University of Stavanger

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

Runtime complexity: $\Theta(V + E)$

Any vertex enters the queue once (V), upon which all its edges are visited. The total number of edges that are visited is E.

Note: sum of adjacent vertices for all vertices is simply |E|

$$\sum_{v \in V} |Adj[v]| = |E|$$

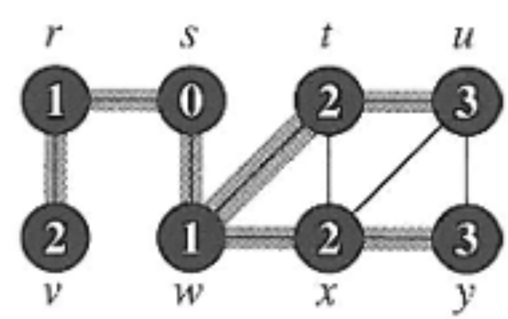
Graph Algorithms (Chapter 22.2)
Breadth Fist Search (BFS)

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

Shortest Path

```
PRINT-PATH(G, s, v)
if v == s
    print s
elseif v.π == NIL
    print "no path from" s "to" v "exists"
else PRINT-PATH(G, s, v.π)
    print v
```

(i)



Q $\emptyset$

University of Stavanger

Claims

Lemma 22.1

Let $G = (V, E)$ be a directed or undirected graph, and let $s \in V$ be an arbitrary vertex. Then, for any edge $(u, v) \in E$,

$$\delta(s, v) \leq \delta(s, u) + 1 .$$

Proof If u is reachable from s, then so is v. In this case, the shortest path from s to v cannot be longer than the shortest path from s to u followed by the edge (u, v), and thus the inequality holds. If u is not reachable from s, then $\delta(s, u) = \infty$, and the inequality holds. ∎

University of Stavanger

Claims

Lemma 22.2

Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. Then upon termination, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies $v.d \geq \delta(s, v)$.

Proof We use induction on the number of ENQUEUE operations. Our inductive hypothesis is that $v.d \geq \delta(s, v)$ for all $v \in V$.

The basis of the induction is the situation immediately after enqueuing s in line 9 of BFS. The inductive hypothesis holds here, because $s.d = 0 = \delta(s, s)$ and $v.d = \infty \geq \delta(s, v)$ for all $v \in V - \{s\}$.

For the inductive step, consider a white vertex v that is discovered during the search from a vertex u. The inductive hypothesis implies that $u.d \geq \delta(s, u)$. From the assignment performed by line 15 and from Lemma 22.1, we obtain

$$\begin{aligned} v.d &= u.d + 1 \\ &\geq \delta(s, u) + 1 \\ &\geq \delta(s, v) . \end{aligned}$$

University of Stavanger

Claims

Lemma 22.3

Suppose that during the execution of BFS on a graph $G = (V, E)$, the queue Q contains the vertices $\langle v_1, v_2, \ldots, v_r \rangle$, where v_1 is the head of Q and v_r is the tail. Then, $v_r.d \leq v_1.d + 1$ and $v_i.d \leq v_{i+1}.d$ for $i = 1, 2, \ldots, r - 1$.

Corollary 22.4

Suppose that vertices v_i and v_j are enqueued during the execution of BFS, and that v_i is enqueued before v_j. Then $v_i.d \leq v_j.d$ at the time that v_j is enqueued.

University of Stavanger

Claims

Theorem 22.5 (Correctness of BFS)

Let $G = (V, E)$ be a directed or undirected graph , and suppose that BFS is run on G from a source vertex $s \in V$. Then, during its execution, BFS discovers every vertex $v \in V$ that is reachable from s, and upon termination $v.d = \delta(s, v)$, $\forall v \in V$. Moreover, for any vertex $v \neq s$ that is reachable from s, one of the shortest paths from s to v is a shortest path from s to $v.\pi$ followed by the edge $(v.\pi, v)$

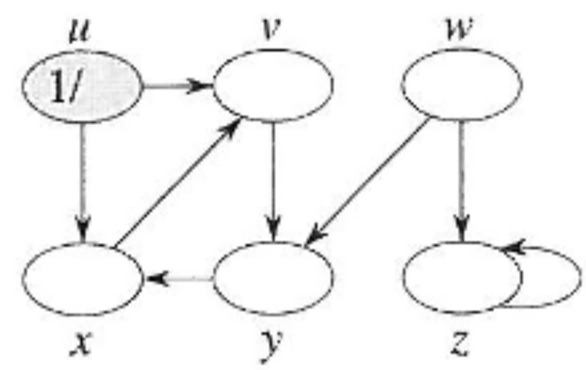
Graph Algorithms (Chapter 22.3)
Depth First Search DFS

Graph Algorithms (Chapter 22.3)
Depth First Search DFS

University of Stavanger

What will be the result of

DFS-VISIT(G,u)?

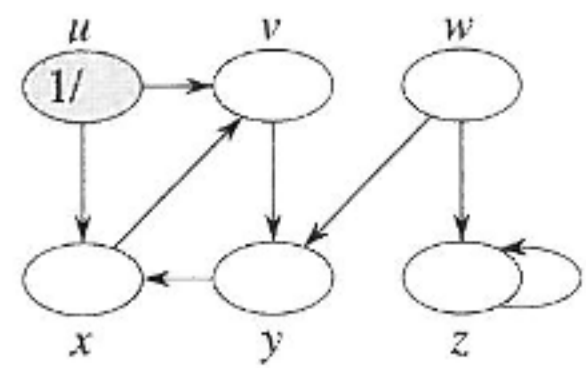


(a)

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

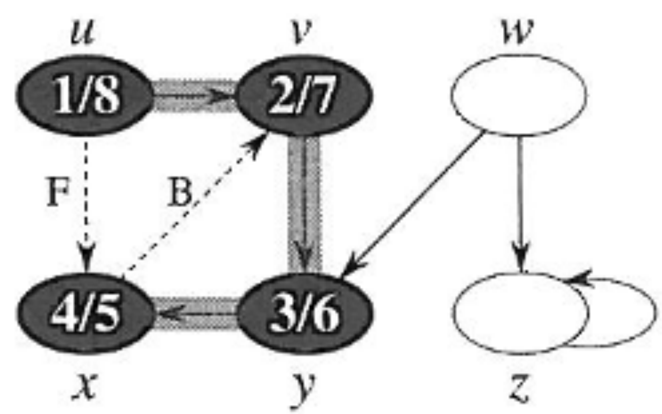
Graph Algorithms (Chapter 22.3)
Depth First Search DFS

But w and z are not discovered, why?



(a)

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

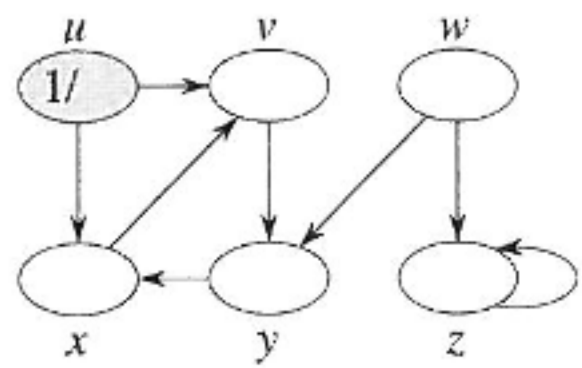


(j)

Graph Algorithms (Chapter 22.3) Depth First Search DFS

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

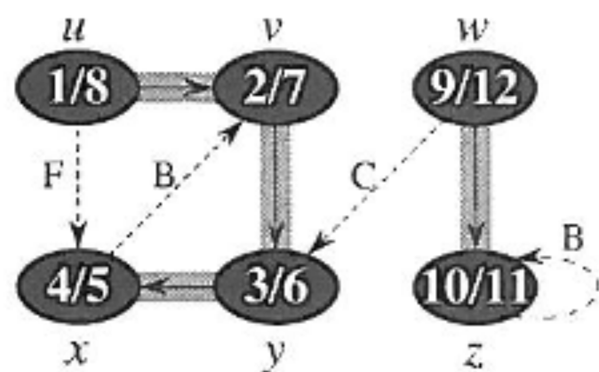
```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```



(a)

The result is a DFS forest

2 trees in this case



(p)

Graph Algorithms (Chapter 22.3) Depth First Search DFS

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

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

Runtime Complexity

All vertices start "white" and every "white" vertex will call VISIT for all its edges. Just like BFS, the runtime of DFS is $\Theta(V + E)$

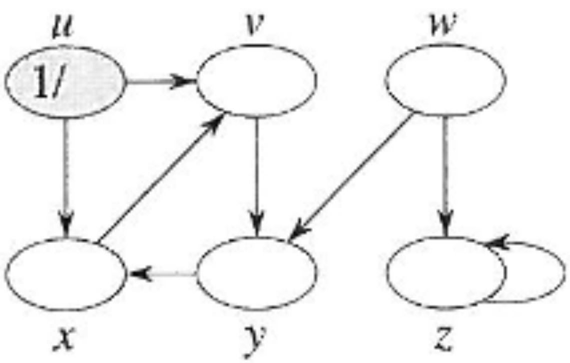
Observation:

$$\sum_{v \in V} |Adj[v]| = |E|$$

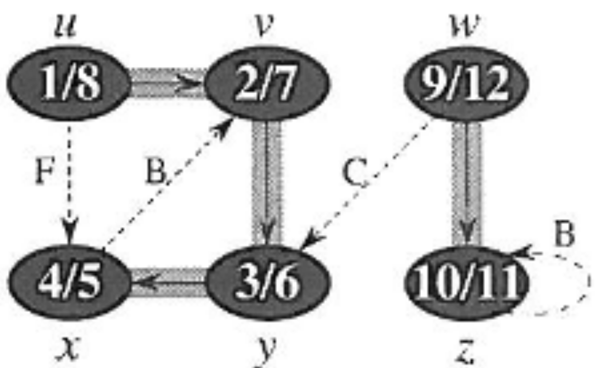
Graph Algorithms (Chapter 22.3) Depth First Search DFS

Type of edges

- <u>Tree edges:</u> example (u,v) because u is parent to v. We have also $u.d < v.d < v.f < u.f$
- <u>Forward edge:</u> example (u,x) because (u,x) not part of the tree, or u is not parent of x. We have also $u.d < x.d < x.f < u.f$ (same!)
- <u>Back edge:</u> example (x,v) because points back to an ancestor, also $v.d \le x.d < x.f \le v.f$
- <u>Cross edge:</u> $y.d < y.f < w.d < w.f$



(a)



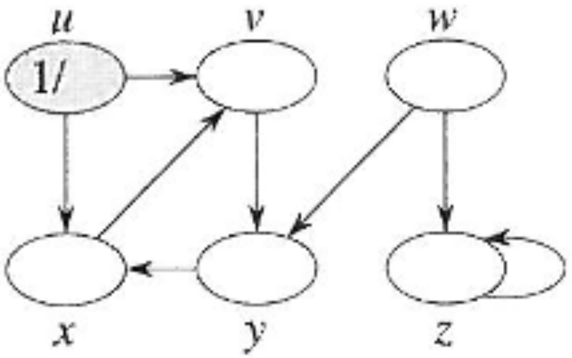
(p)

Graph Algorithms (Chapter 22.3) Depth First Search DFS

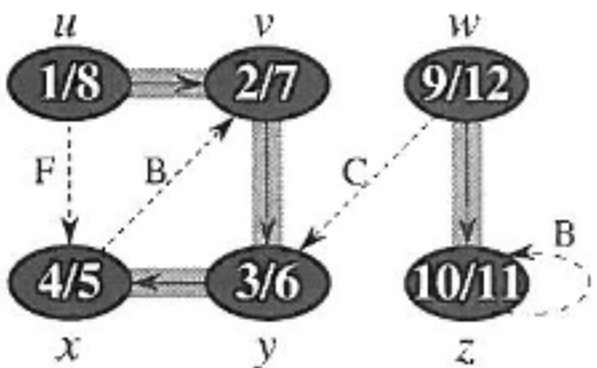
Important!

Back edge => The graph is cyclic

- <u>Back edge: </u>example (x,v) because points back to an ancestor, also $v.d \le x.d < x.f \le v.f$



(a)



(p)

Graph Algorithms (Chapter 22.3) Depth First Search DFS


```
DFS(G)
for each vertex u ∈ G.V
        u.color = WHITE
        u.π = NIL
time = 0
for each vertex u ∈ G.V
        if u.color == WHITE
                DFS-VISIT(G, u)
```

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

DFS information on edges

White=> tree edge

Grey=> back edge

Black=>forward or cross edge

Graph Algorithms (Chapter 22.4)
Topological sort

A topological sort of a graph is a direct application of DFS !

Graph Algorithms (Chapter 22.4) Topological sort

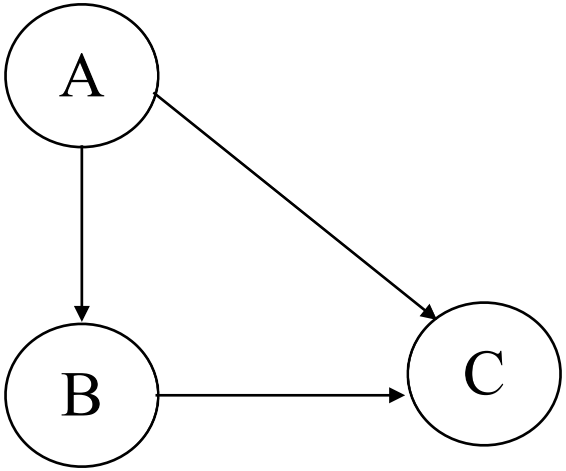
Definition

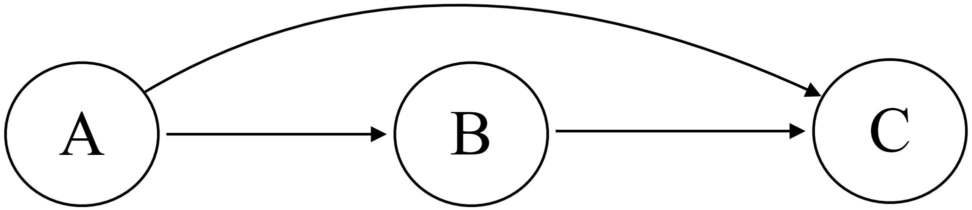
A topological sort of a Directed Acyclic Graph (DAG) $G = (V, E)$ is a linear ordering of all its vertices such that if $(u, v) \in E$, then u appears before v in the linear ordering

Graph Algorithms (Chapter 22.4) Topological sort

University of Stavanger

Example

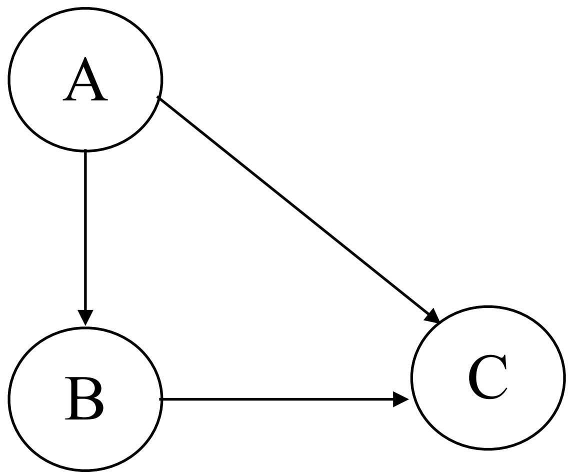


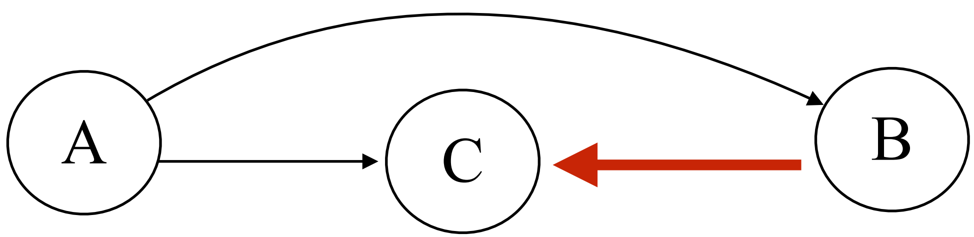


Is a topological order

Graph Algorithms (Chapter 22.4)
Topological sort

Example

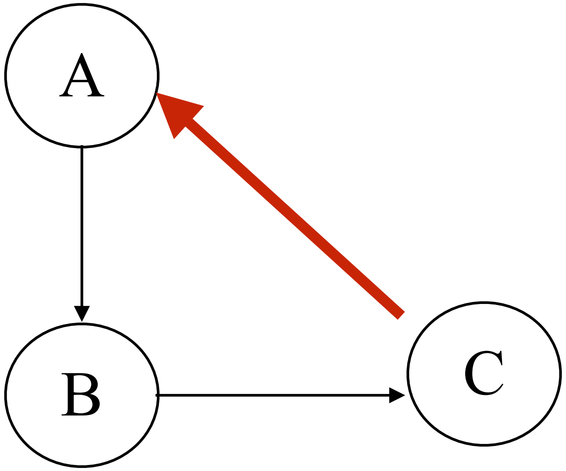




Is NOT a topological order

Graph Algorithms (Chapter 22.4) Topological sort

Example



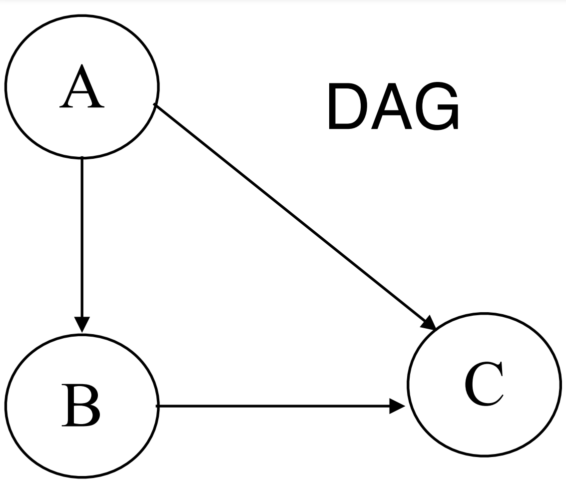
IS NOT A DAG => No Topological order

Graph Algorithms (Chapter 22.4) Topological sort

Example

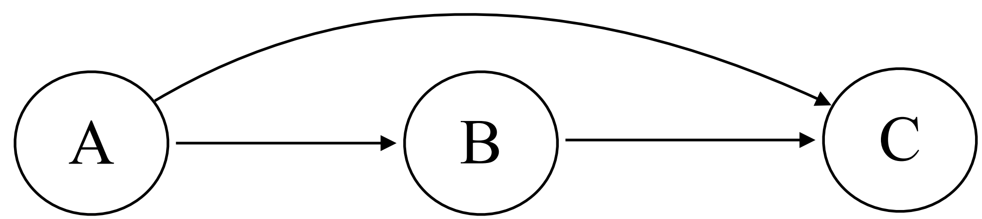
Lets apply to the algorithm on the DAG to find a topological order.

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



DAG

Topological Order

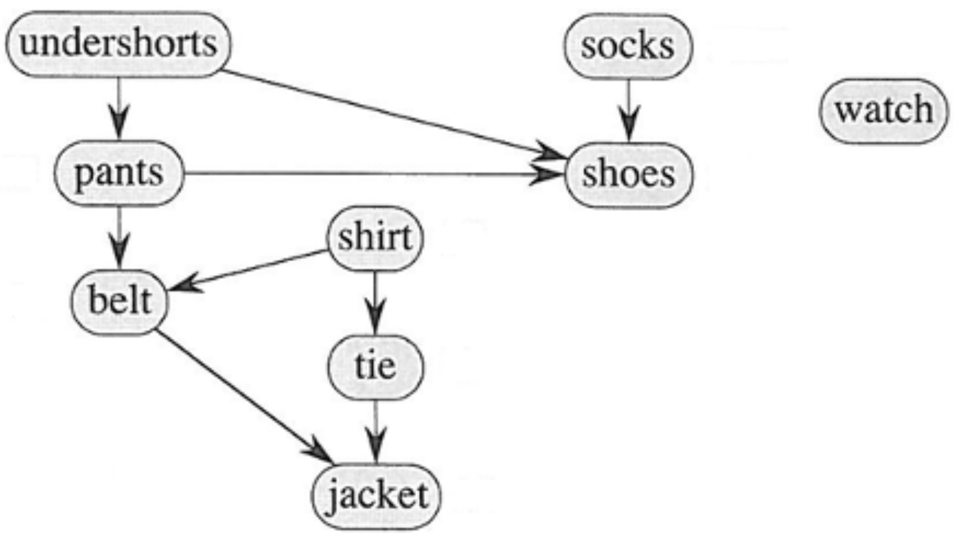


Graph Algorithms (Chapter 22.4)
Topological sort

Example

TOPOLOGICAL-SORT(G)

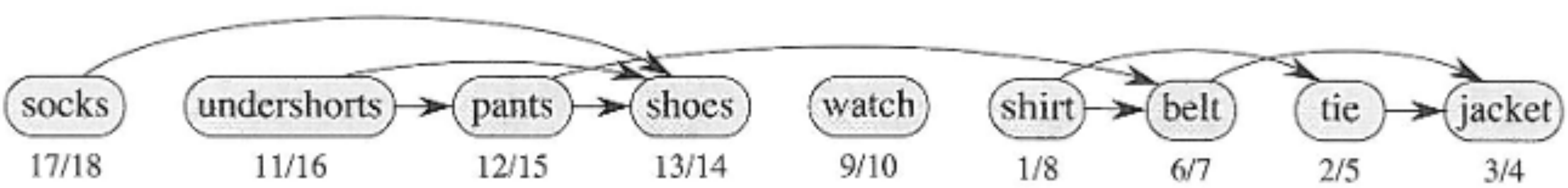
1 call DFS(G) to compute finishing times $v.f$ for each vertex v
2 as each vertex is finished, insert it onto the front of a linked list
3 **return** the linked list of vertices



Example

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



University of Stavanger

Runtime complexity

$$\Theta(V + E)$$

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

Graph Algorithms (Chapter 22.4)
Topological sort

Given a graph how would you do to determine if it has a topological sort ?

Graph Algorithms (Chapter 22.4) Topological sort

Given a graph how would you do to determine if it has a topological sort ?

We have to find out if it is a DAG

Lemma 22.11
A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

Recall that during DFS if we meet a grey vertex => back edge

Graph Algorithms (Chapter 22.4)
Topological sort


```
DFS(G)
for each vertex u ∈ G.V
        u.color = WHITE
        u.π = NIL
time = 0
for each vertex u ∈ G.V
        if u.color == WHITE
                DFS-VISIT(G, u)
```

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

Modify this algorithm to detect back edges

Graph Algorithms (Chapter 22.5)
Strongly Connected Components (SCC)

University of Stavanger

What is a strongly connected component?

Strongly Connected Components (SCC)

A strongly connected component of a directed graph $G = (V, E)$ is a maximum set of vertices $C \subseteq V$ such that every pair u and v we have a path from u to v, and a path from v to u.

We need an algorithm that computes all the strongly connected components of a directed graph.

Graph Algorithms (Chapter 22.5) Strongly Connected Components (SCC)

University of Stavanger

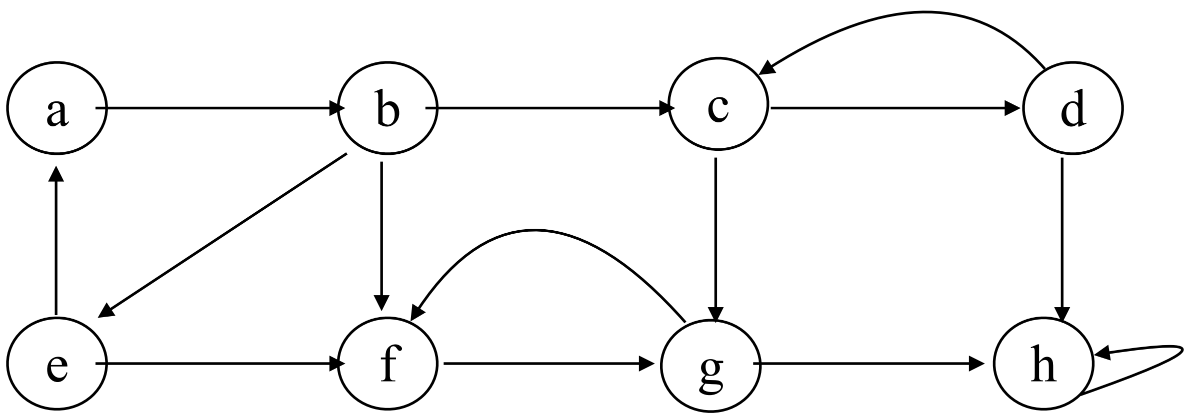
A strongly connected component of a directed graph $G = (V, E)$ is a maximum set of vertices $C \subseteq V$ such that every pair u and v we have a path from u to v, and a path from v to u.

```
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
```

Graph Algorithms (Chapter 22.5) Strongly Connected Components (SCC)

University of Stavanger

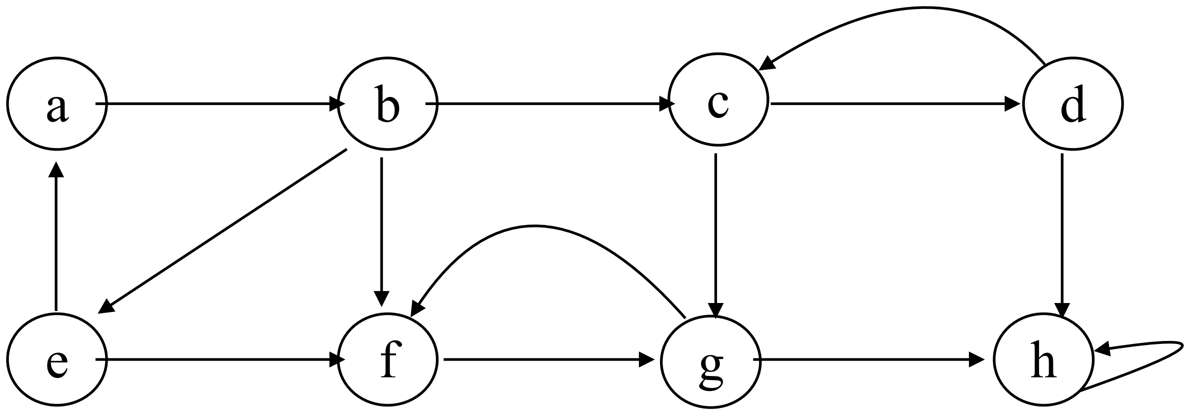
Exercise: Find the strongly connected components of this graph

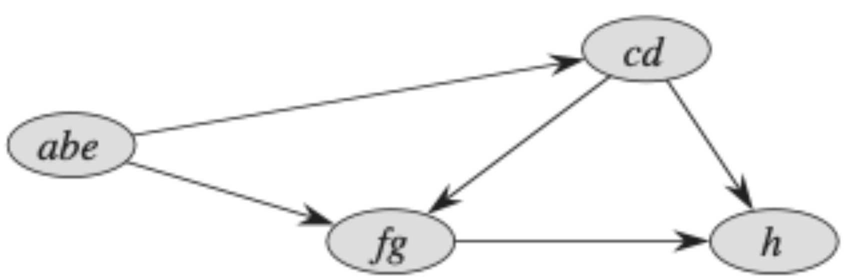


STRONGLY-CONNECTED-COMPONENTS(G)

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

Exercise: Find the strongly connected components of this graph





The result is a DAG

Summary Chapter 22

- Graphs can be represented as adjacency list or matrix
- BFS marks every vertex with its shortest distance to a source
- The shortest distance from s to v is the shortest distance from s to $v.\pi + 1$
- DFS does not provide to the shortest path from a source s to other vertices but rather answer reachability
- DFS marks discovery and finish time which in turn provide interesting properties
- DFS stands behind topology sorting, and finding strongly connected components of a graph

Exercices

Will come