IDMA 2025: WEEK 6

GRAPH TRAVERSAL ALGORITHMS: BFS & DFS

SRIKANTH SRINIVASAN

Algorithms & Complexity Section

DIKU

srsr@di.ku.dk

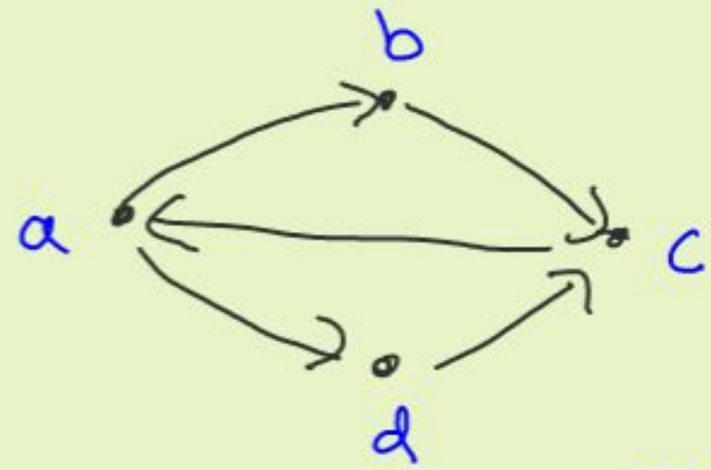
Graphs

Directed graph

$G = (V, E)$

V - vertices

E - edges, each edge an ordered pair (u, v)

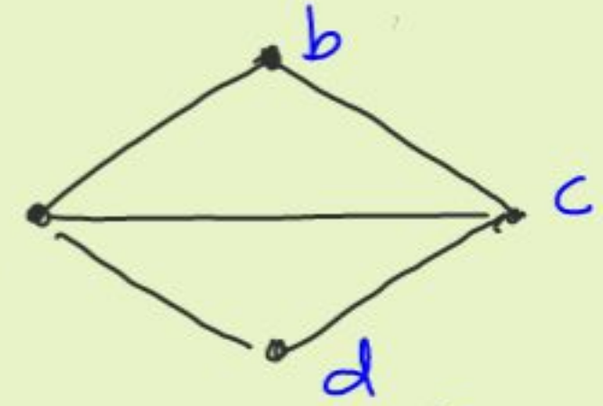


$V = \{a, b, c, d\}$

$E = \{(a,b), (b,c), (c,a), (a,d), (d,c)\}$

Undirected graph

E - each edge an "unordered pair", ie a set of two vertices



$V = \{a, b, c, d\}$

$E = \{\{a,b\}, \{b,c\}, \{a,c\}, \{a,a\}, \{d,c\}\}$

Neighbours, degree

Out-neighbours of $u \in V$

$$N_{out}(u) := \{v \mid (u,v) \in E\}$$

Out degree

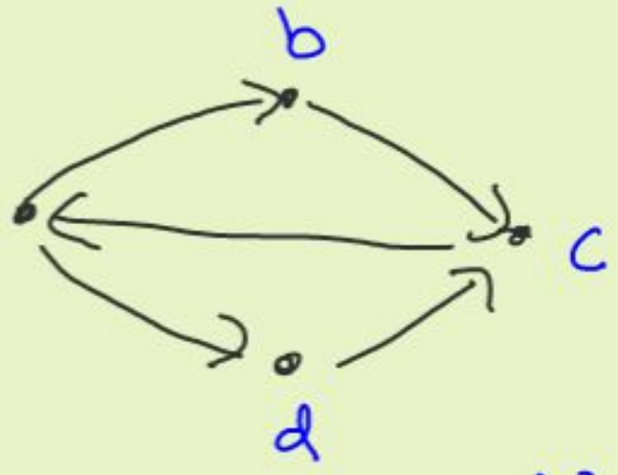
$$deg_{out}(u) = |N_{out}(u)|$$

In-neighbours

$$N_{in}(u) = \{v \mid (v,u) \in E\}$$

In-degree

$$deg_{in}(u) = |N_{in}(u)|$$



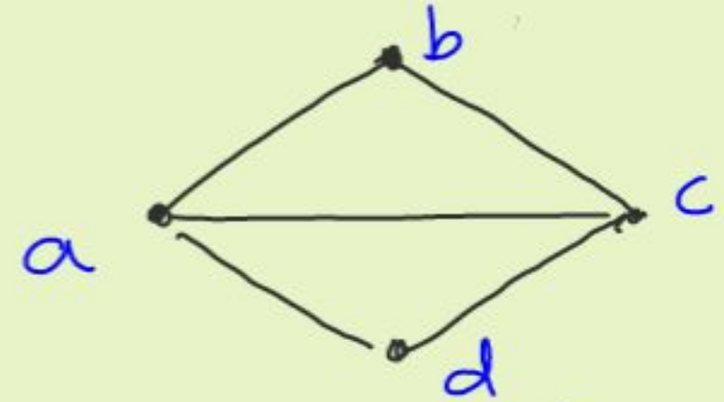
$V = \{a, b, c, d\}$

$E = \{(a,b), (b,c), (c,a), (a,d), (d,c)\}$

Neighbours of $u \in V$

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

Degree $deg(u) = |N(u)|$



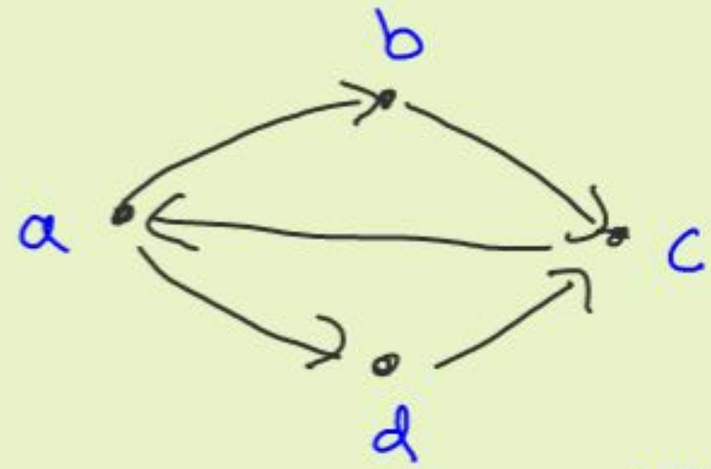
$V = \{a, b, c, d\}$

$E = \{\{a,b\}, \{b,c\}, \{a,c\}, \{a,a\}, \{d,c\}\}$

Neighbours, degree

$$\sum_{u \in V} deg_{out}(u) = |E|$$

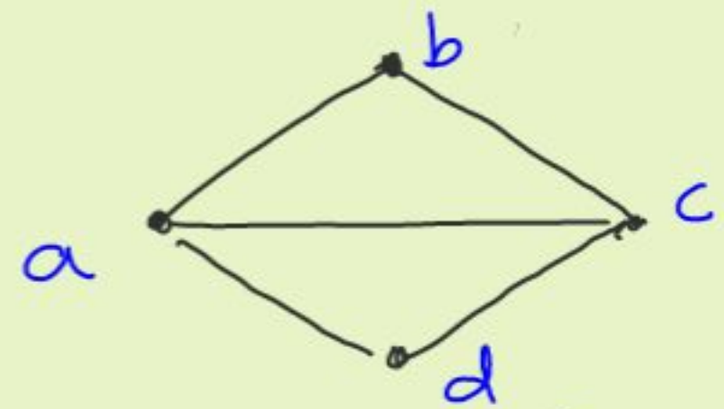
$$\sum_{u \in V} deg_{in}(u) = |E|$$



$V = \{a, b, c, d\}$

$E = \{(a,b), (b,c), (c,a), (a,d), (d,c)\}$

$$\frac{1}{2} \sum_{u \in V} deg(u) = |E|$$



$V = \{a, b, c, d\}$

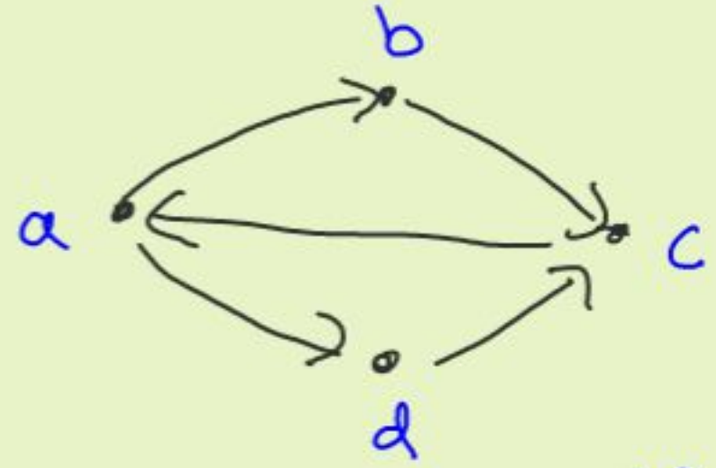
$E = \{\{a,b\}, \{b,c\}, \{a,c\}, \{a,a\}, \{d,c\}\}$

Paths & Simple Paths

Path - sequence of vertices

$(v_0, v_1, \ldots, v_n)$ s.t.

$\forall i \in \{0, \ldots, n-1\} : (v_i, v_{i+1}) \in E$



$V = \{a, b, c, d\}$

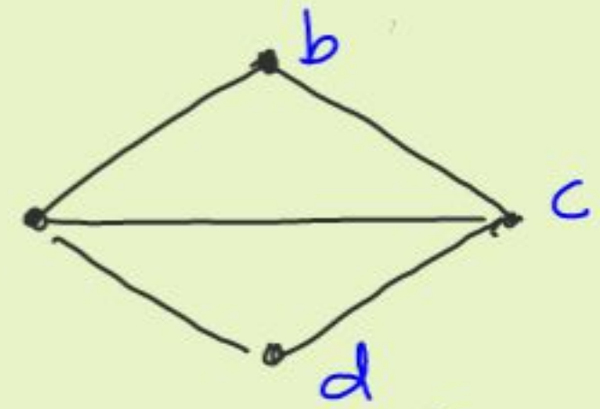
$E = \{(a,b), (b,c), (c,a), (a,d), (d,c)\}$

Simple Path - vertices are not repeated

(a, b, c, d) - Simple path

(a, b, c, a, d) - Path, not simple

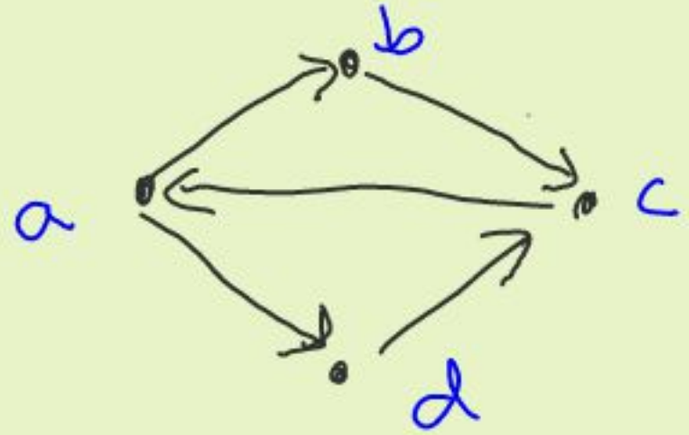
Cycle - $v_0 = v_n$ & no repeated edges.



$V = \{a, b, c, d\}$

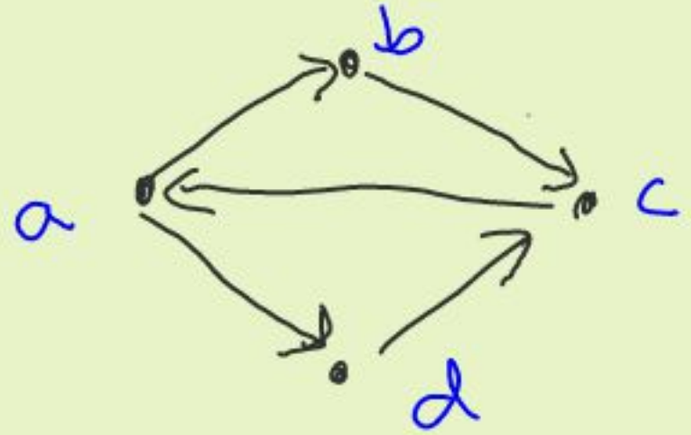
$E = \{\{a,b\}, \{b,c\}, \{a,c\}, \{a,a\}, \{d,c\}\}$

Representing a graph



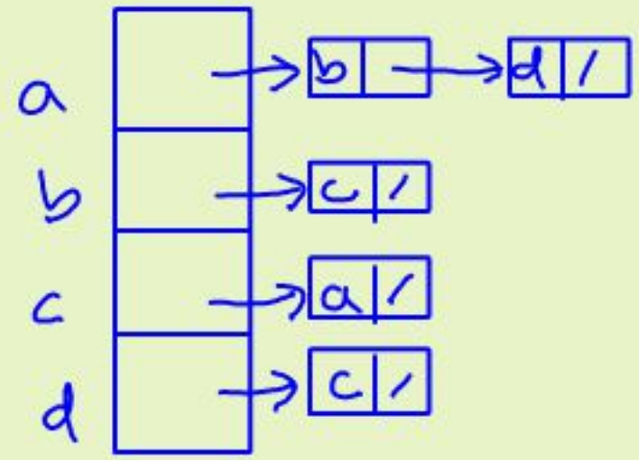
Adjacency matrix Adjacency list

Representing a graph



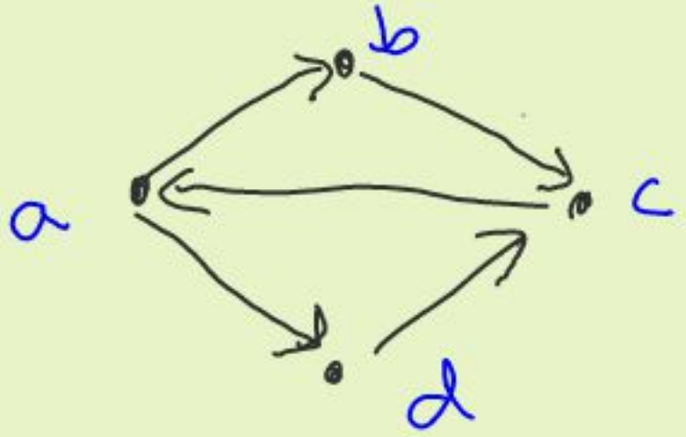
	a	b	c	d
a	0	1	0	1
b	0	0	1	0
c	1	0	0	0
d	0	0	1	0

Adjacency matrix



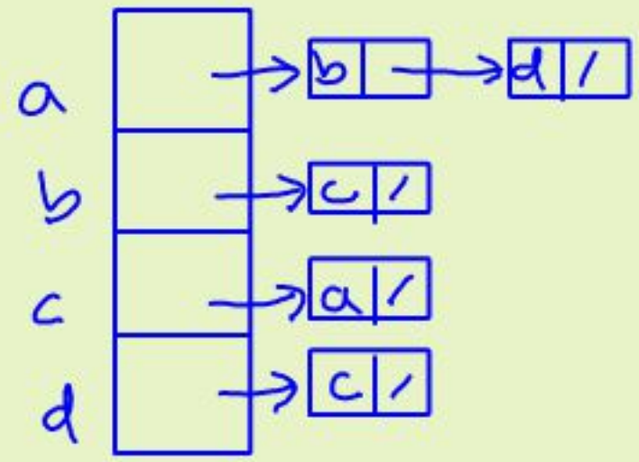
Adjacency List

Representing a graph



	a	b	c	d
a	0	1	0	1
b	0	0	1	0
c	1	0	0	0
d	0	0	1	0

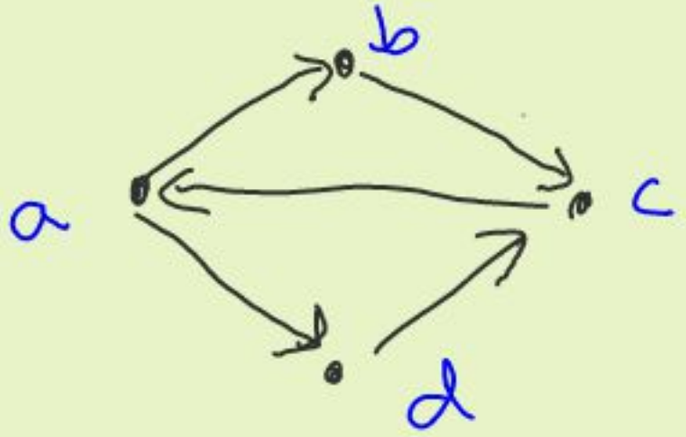
Adjacency matrix



Adjacency List

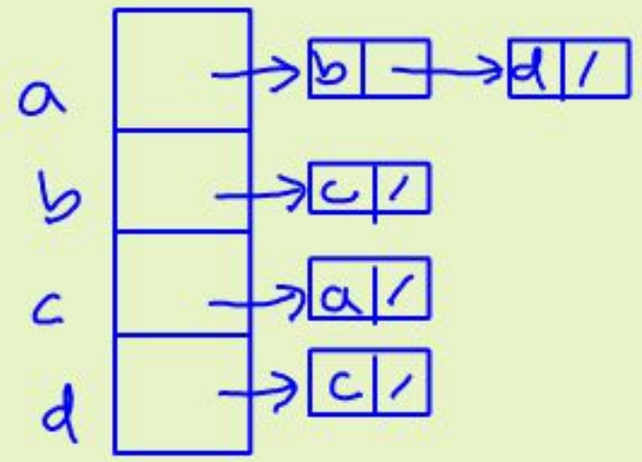
	Adjacency matrix	Adjacency list
Checking if $(u,v) \in E$.		
List all out-nbrs. of u.		

Representing a graph



	a	b	c	d
a	0	1	0	1
b	0	0	1	0
c	1	0	0	0
d	0	0	1	0

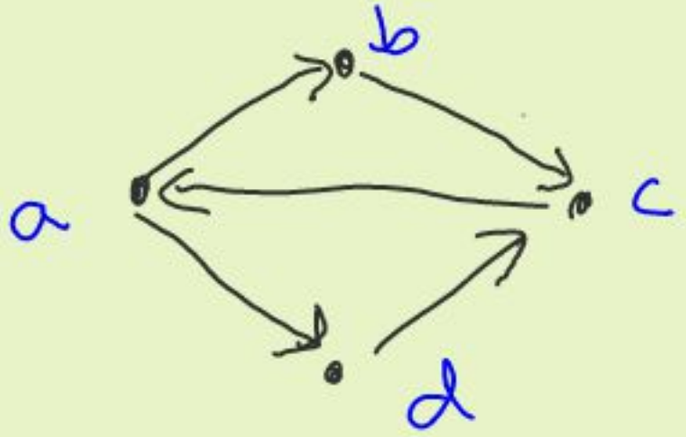
Adjacency matrix



Adjacency list

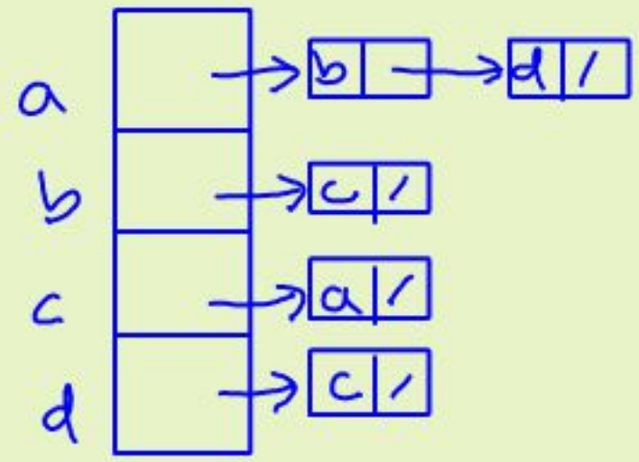
	Adjacency matrix	Adjacency list
Checking if $(u,v) \in E$.	$O(1)$	
List all out-nbrs. of u.	$O(\|V\|)$	

Representing a graph



	a	b	c	d
a	0	1	0	1
b	0	0	1	0
c	1	0	0	0
d	0	0	1	0

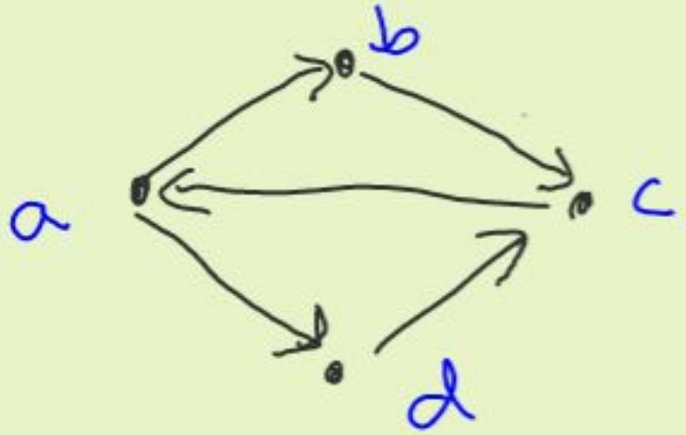
Adjacency matrix



Adjacency List

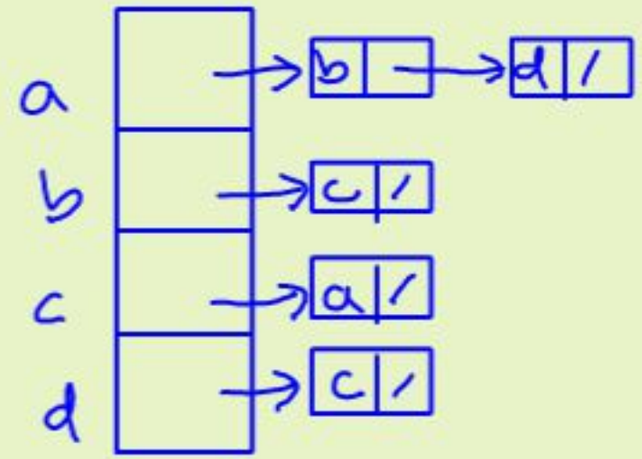
	Adjacency matrix	Adjacency list
Checking if $(u,v) \in E$.	$O(1)$	$O(deg_{out}(u))$
List all out-nbrs. of u.	$O(\|V\|)$	$O(deg_{out}(u))$

Representing a graph



	a	b	c	d
a	0	1	0	1
b	0	0	1	0
c	1	0	0	0
d	0	0	1	0

Adjacency matrix



Adjacency List

	Adjacency matrix	Adjacency list
Checking if $(u,v) \in E$.	$O(1)$	$O(\deg_{out}(u))$
List all out-nbrs. of u.	$O(\|V\|)$	$O(\deg_{out}(u))$

Pros & Cons of Adj. matrix

Pro: Some operations faster

Con: Space inefficient

Usually, adj. lists are better.

Graph Traversal

Input : $G = (V, E)$ directed / undirected

Aim : Explore the graph to understand its structure.

Graph Traversal

Input : $G = (V, E)$ directed / undirected

Aim : Explore the graph to understand its structure.

Applications : connectivity, distances, ordering...

Graph Traversal

Input: $G = (V, E)$ directed / undirected

Aim: Explore the graph to understand its structure.

Applications: connectivity, distances, ordering...

Two algorithms:

- Breadth-First Search (BFS)
- Depth-First Search (DFS)

Breadth-first Search (BFS)

Breadth-first Search (BFS)

→ Start vertex s

→ Get list of out-neighbours & add to queue Q

→ Visit vertices in queue in order from Q.

→ Stop when Q empty.

Breadth-first Search (BFS)

→ Start vertex s

→ Get list of out-neighbours & add to queue Q

→ Visit vertices in queue in order from Q.

→ Stop when Q empty.

BFS(G, s)

```
(1)  for v ∈ V(G)
(2)      v.discovered = FALSE
(3)  s.discovered = TRUE
(4)  s.dist = 0
(5)  Q.enqueue(s)
(6)  while NOT(Q.empty())
(7)      v = Q.dequeue()
(8)      for w ∈ N(v)
(9)          if NOT(w.discovered)
(10)             w.discovered = TRUE
(11)             w.dist = v.dist + 1
(12)             w.pred = v
(13)             Q.enqueue(w)
```

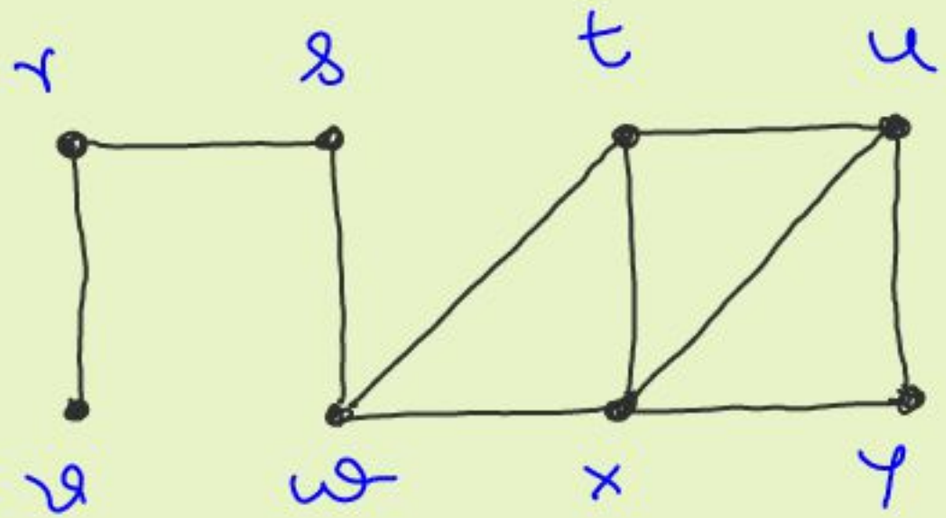
Breadth-first Search (BFS)

→ Start vertex s

→ Get list of out-neighbours & add to queue Q

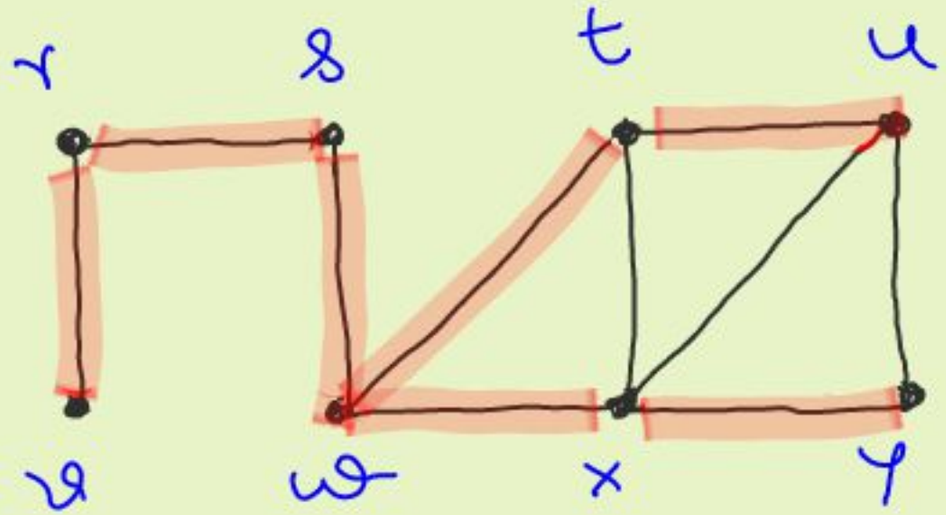
→ Visit vertices in queue in order from Q.

→ Stop when Q empty.




```
BFS(G, s)
(1)  for v ∈ V(G)
(2)      v.discovered = FALSE
(3)  s.discovered = TRUE
(4)  s.dist = 0
(5)  Q.enqueue(s)
(6)  while NOT(Q.empty())
(7)      v = Q.dequeue()
(8)      for w ∈ N(v)
(9)          if NOT(w.discovered)
(10)             w.discovered = TRUE
(11)             w.dist = v.dist + 1
(12)             w.pred = v
(13)             Q.enqueue(w)
```

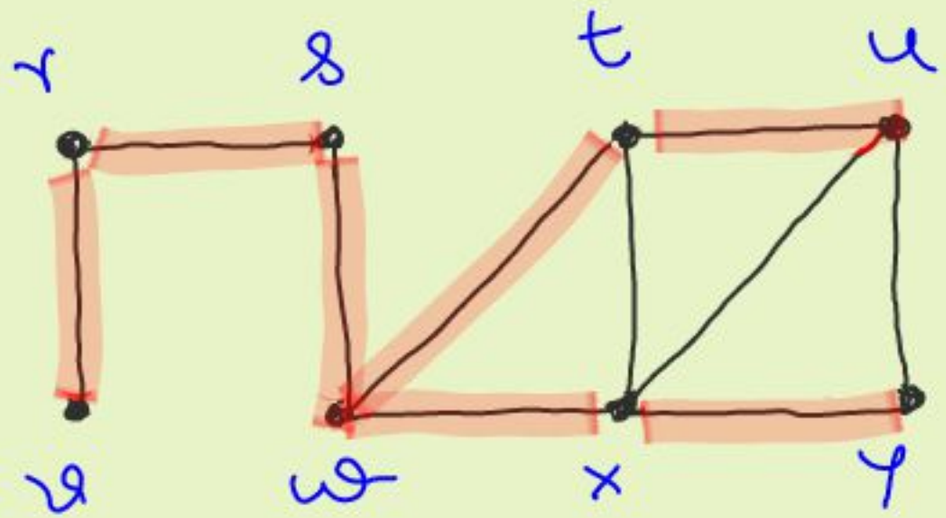
Breadth-first Search (BFS)




```
BFS(G,s)
(1)  for v∈V(G)
(2)      v.discovered = FALSE
(3)  s.discovered = TRUE
(4)  s.dist = 0
(5)  Q.enqueue(s)
(6)  while NOT(Q.empty())
(7)      v = Q.dequeue()
(8)      for w ∈ N(v)
(9)          if NOT(w.discovered)
(10)             w.discovered = TRUE
(11)             w.dist = v.dist + 1
(12)             w.pred = v
(13)             Q.enqueue(w)
```

Breadth-first Search (BFS)

Properties of BFS

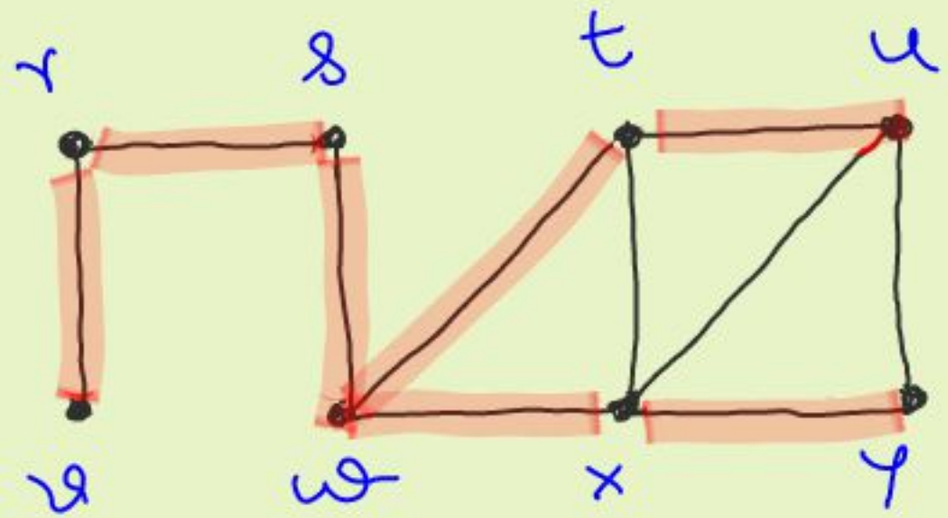



```
BFS(G,s)
(1)  for v∈V(G)
(2)      v.discovered = FALSE
(3)  s.discovered = TRUE
(4)  s.dist = 0
(5)  Q.enqueue(s)
(6)  while NOT(Q.empty())
(7)      v = Q.dequeue()
(8)      for w ∈ N(v)
(9)          if NOT(w.discovered)
(10)             w.discovered = TRUE
(11)             w.dist = v.dist + 1
(12)             w.pred = v
(13)             Q.enqueue(w)
```

Breadth-first Search (BFS)

Properties of BFS

→ Finds shortest path from s to each vertex reachable from s.




```
BFS(G, s)
(1)  for v ∈ V(G)
(2)      v.discovered = FALSE
(3)  s.discovered = TRUE
(4)  s.dist = 0
(5)  Q.enqueue(s)
(6)  while NOT(Q.empty())
(7)      v = Q.dequeue()
(8)      for w ∈ N(v)
(9)          if NOT(w.discovered)
(10)             w.discovered = TRUE
(11)             w.dist = v.dist + 1
(12)             w.pred = v
(13)             Q.enqueue(w)
```

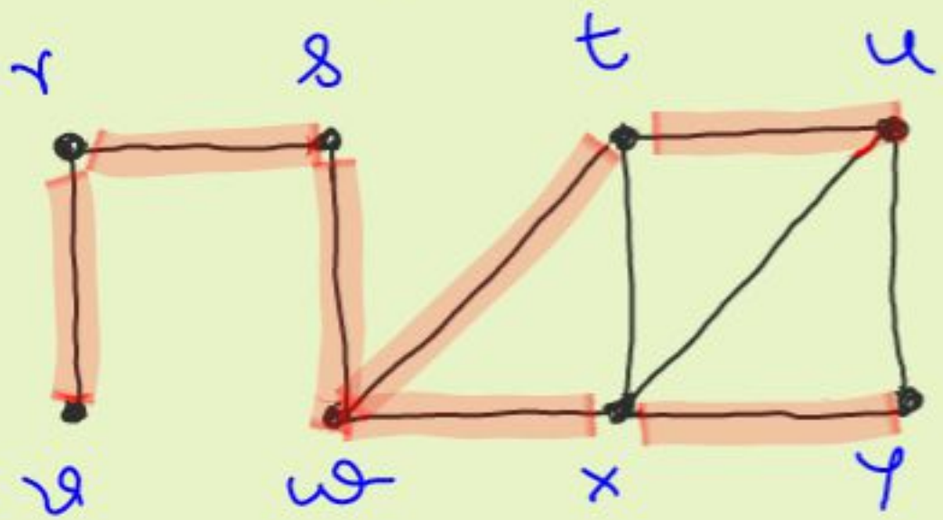
Breadth-first Search (BFS)

Properties of BFS

→ Finds shortest path from s to each vertex reachable from s.

→ G undirected, connected

⇒ predecessor edges form a spanning tree.




```
BFS(G,s)
(1)  for v∈V(G)
(2)      v.discovered = FALSE
(3)  s.discovered = TRUE
(4)  s.dist = 0
(5)  Q.enqueue(s)
(6)  while NOT(Q.empty())
(7)      v = Q.dequeue()
(8)      for w ∈ N(v)
(9)          if NOT(w.discovered)
(10)             w.discovered = TRUE
(11)             w.dist = v.dist + 1
(12)             w.pred = v
(13)             Q.enqueue(w)
```

Breadth-first Search (BFS)

Running time

```
BFS(G,s)
O(|V|) { (1) for v∈V(G)
         (2)     v.discovered := FALSE
O(1)   { (3) s.discovered := TRUE
         (4) s.dist := 0
         (5) Q.enqueue(s)
(6) while NOT(Q.empty())
(7)     v := Q.dequeue()
(8)     for w ∈ N(v)
(9)         if NOT(w.discovered)
(10)            w.discovered: TRUE
(11)            w.dist := v.dist + 1
(12)            w.pred := v
(13)            Q:enqueue(w)
```

Breadth-first Search (BFS)

Running time

(assuming adjacency lists)

```
BFS(G,s)
O(|V|) {  (1) for v∈V(G)
          (2)     v.discovered := FALSE
O(1)   {  (3) s.discovered := TRUE
          (4) s.dist := 0
          (5) Q.enqueue(s)
          (6) while NOT(Q.empty())
          (7)     v := Q.dequeue()
          (8)     for w ∈ N(v)
          (9)         if NOT(w.discovered)
          (10)            w.discovered := TRUE
          (11)            w.dist := v.dist + 1
          (12)            w.pred := v
          (13)            Q.enqueue(w)
```

O(deg_out(v))

Breadth-first Search (BFS)

BFS(G, s)

O(|V|) {
① For v ∈ V(G)
② v.discovered := FALSE
}
O(1) {
③ s.discovered := TRUE
④ s.dist := 0
⑤ Q.enqueue(s)
}
⑥ while NOT(Q.empty())
⑦ v := Q.dequeue()
⑧ for w ∈ N(v)
⑨ if NOT(w.discovered)
⑩ w.discovered: TRUE
⑪ w.dist := v.dist + 1
⑫ w.pred := v
⑬ Q.enqueue(w)

O(deg_out(v)) (⑧–⑬)

Running time

$O(|V|) + O(1) + c \cdot \sum_{v \in V} deg_{out}(v)$

(assuming adjacency lists)

Breadth-first Search (BFS)

BFS(G, s)

```
O(|V|)  { (1) for v ∈ V(G)
          (2)     v.discovered := FALSE
O(1)    { (3) s.discovered := TRUE
          (4) s.dist := 0
          (5) Q.enqueue(s)
          (6) while NOT(Q.empty())
          (7)     v := Q.dequeue()
          (8)     for w ∈ N(v)
          (9)         if NOT(w.discovered)
          (10)            w.discovered: TRUE
          (11)            w.dist := v.dist + 1
          (12)            w.pred := v
          (13)            Q.enqueue(w)
```

O(deg_out(v)) (lines 8–13)

Running time

$O(|V|) + O(1)$

$+ c \cdot \sum_{v \in V} deg_{out}(v)$

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

(assuming adjacency lists)

Depth-first Search (DFS)

→ Recursive algorithm

→ Tries to explore "deeper" whenever possible.

Depth-first Search (DFS)

→ Recursive algorithm

→ Tries to explore "deeper" whenever possible.

v.dtime discovery time
v.ftime finish time
v.pred predecessor
v.colour (W/G/B)
time (global variable)

Depth-first Search (DFS)

→ Recursive algorithm

→ Tries to explore "deeper" whenever possible.

$v.dtime$ discovery time
$v.ftime$ finish time
$v.pred$ predecessor
$v.colour$ (W/G/B)
time (global variable)

DFS-visit (G, v)

```
(1)  v.dtime := time
(2)  time := time + 1
(3)  v.colour := G
(4)  For all w ∈ N(v)
(5)      if (w.colour == W)
(6)          w.pred := v
(7)          DFS-visit (G, w)
(8)  v.colour := B
(9)  v.ftime := time
(10) time := time + 1.
```

Depth-first Search (DFS)

→ Recursive algorithm

→ Tries to explore "deeper" whenever possible.

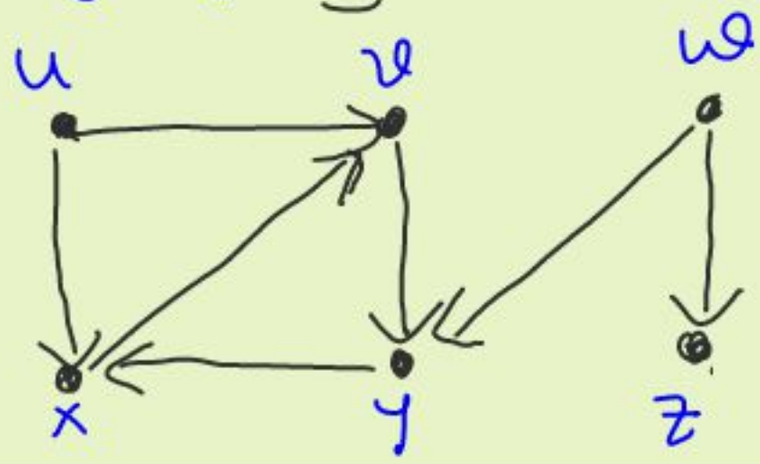
$v.dtime$ discovery time

$v.ftime$ finish time

$v.pred$ predecessor

$v.colour$ (W/G/B)

time (global variable)



DFS-visit (G,v)

(1) $v.dtime := time$
(2) $time := time + 1$
(3) $v.colour := G$
(4) For all $w \in N(v)$
(5) if ($w.colour == W$)
(6) $w.pred := v$
(7) DFS-visit (G,w)
(8) $v.colour := B$
(9) $v.ftime := time$
(10) $time := time + 1$.

Depth-first Search (DFS)

→ Recursive algorithm

→ Tries to explore "deeper" whenever possible.

DFS(G)

1. For each $v \in V(G)$
2. $\quad v.colour := W$
3. time := 0
4. for each $v \in V(G)$
5. $\quad$ if ($v.colour == W$)
6. $\quad\quad$ DFS-visit (G, v)

DFS-visit (G, v)

1. $v.dtime := time$
2. $time := time + 1$
3. $v.colour := G$
4. For all $w \in N(v)$
5. $\quad$ if ($w.colour == W$)
6. $\quad\quad w.pred := v$
7. $\quad\quad$ DFS-visit ((G, w)
8. $v.colour := B$
9. $v.ftime := time$
10. $time := time + 1$.

Depth-first Search (DFS)

→ Recursive algorithm

→ Tries to explore "deeper" whenever possible.

DFS(G)

1. For each $v \in V(G)$
2. $\quad v.colour := W$
3. time := 0
4. for each $v \in V(G)$
5. $\quad$ if ($v.colour == W$)
6. $\quad\quad$ DFS-visit (G, v)

Running time

DFS-visit (G, v)

1. $v.dtime := time$
2. $time := time + 1$
3. $v.colour := G$
4. For all $w \in N(v)$
5. $\quad$ if ($w.colour == W$)
6. $\quad\quad w.pred := v$
7. $\quad\quad$ DFS-visit ((G, w)
8. $v.colour := B$
9. $v.ftime := time$
10. $time := time + 1$.

Depth-first Search (DFS)

→ Recursive algorithm

→ Tries to explore "deeper" whenever possible.

DFS(G)

1. For each $v \in V(G)$
2. $\quad v.colour := W$
3. time := 0
4. for each $v \in V(G)$
5. $\quad$ if ($v.colour == W$)
6. $\quad\quad$ DFS-visit (G, v)

Running time $O(|V|) + c\sum_{v} deg_{out}(v)$

DFS-visit (G, v)

1. $v.dtime := time$
2. $time := time + 1$
3. $v.colour := G$
4. For all $w \in N(v)$
5. $\quad$ if ($w.colour == W$)
6. $\quad\quad w.pred := v$
7. $\quad\quad$ DFS-visit ((G, w)
8. $v.colour := B$
9. $v.ftime := time$
10. $time := time + 1$.

Depth-first Search (DFS)

→ Recursive algorithm

→ Tries to explore "deeper" whenever possible.

DFS(G)

1. For each $v \in V(G)$
2. $\quad v.colour := W$
3. time := 0
4. for each $v \in V(G)$
5. $\quad$ if ($v.colour == W$)
6. $\quad\quad$ DFS-visit (G, v)

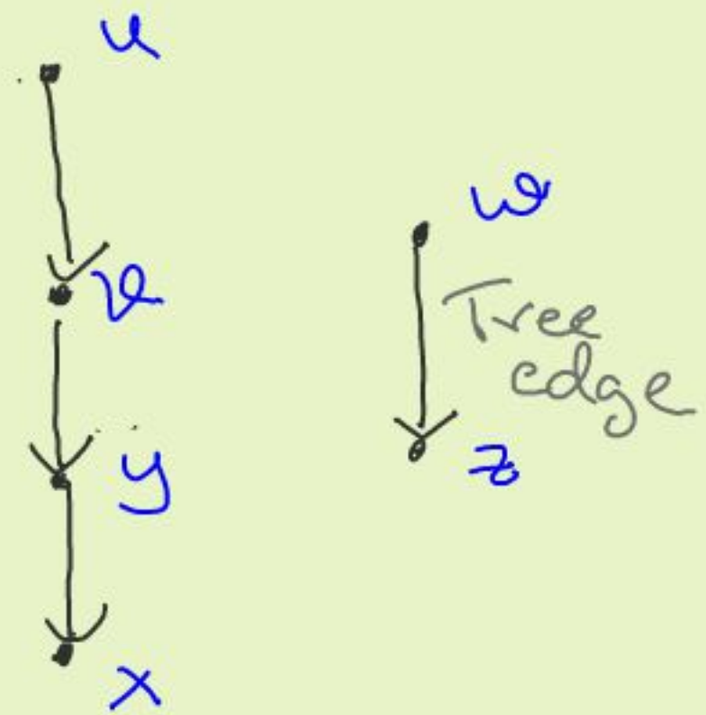
DFS-visit (G, v)

1. $v.dtime := time$
2. $time := time + 1$
3. $v.colour := G$
4. For all $w \in N(v)$
5. $\quad$ if ($w.colour == W$)
6. $\quad\quad w.pred := v$
7. $\quad\quad$ DFS-visit (G, w)
8. $v.colour := B$
9. $v.ftime := time$
10. $time := time + 1$.

Running time $O(|V|) + c \sum_{v} deg_{out}(v) = O(|V|+|E|)$

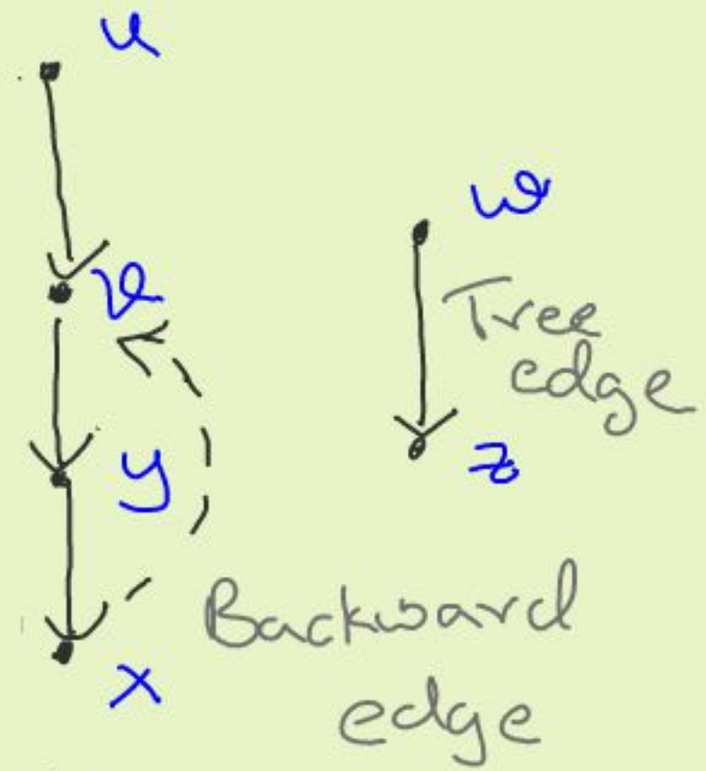
DFS forest

DFS forest : made up
of edges from
the predecessor of a vertex
to a vertex (Tree edges)



DFS forest

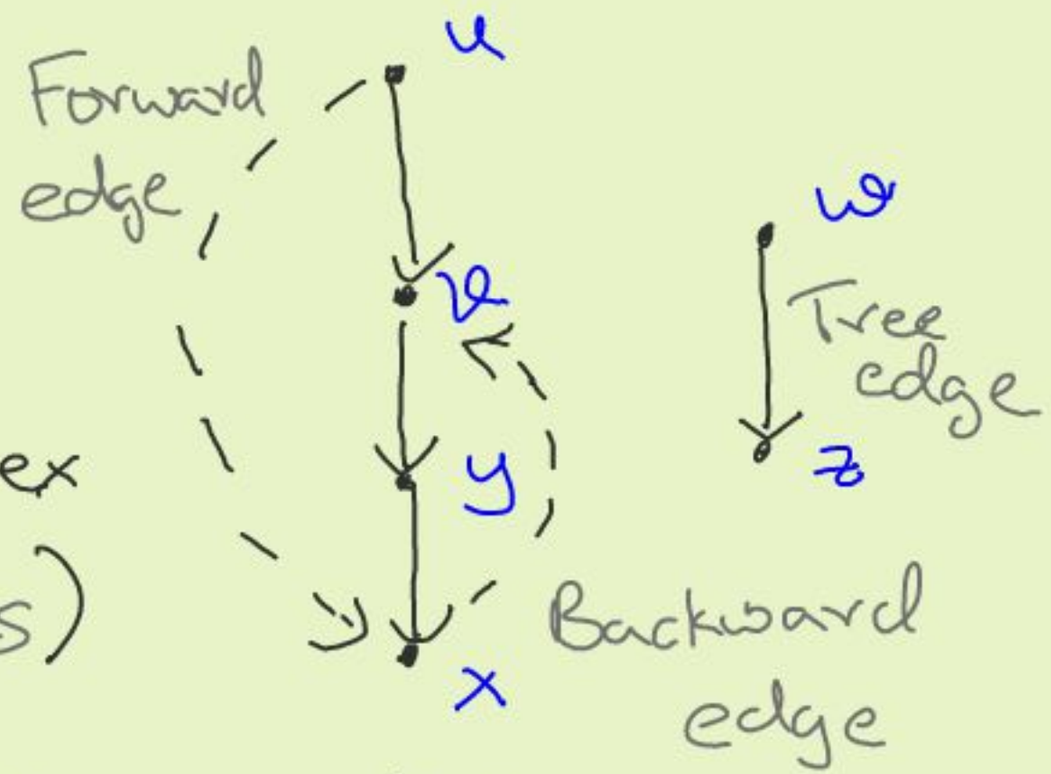
DFS forest : made up
of edges from
the predecessor of a vertex
to a vertex (Tree edges)



Back edge : vertex to its ancestor
in DFS forest.

DFS forest

DFS forest : made up of edges from the predecessor of a vertex to a vertex (Tree edges)

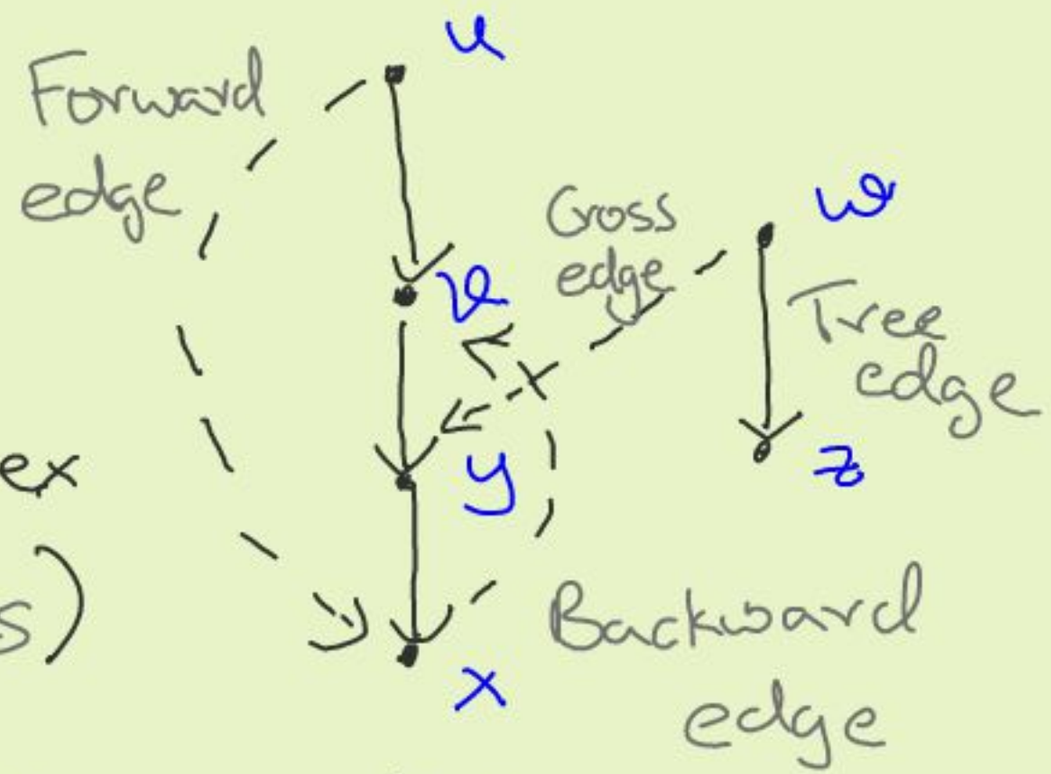


Back edge : vertex to its ancestor in DFS forest.

Forward edge : vertex to descendant in DFS forest

DFS forest

DFS forest : made up of edges from the predecessor of a vertex to a vertex (Tree edges)



Back edge : vertex to its ancestor in DFS forest.

Forward edge : vertex to descendant in DFS forest

Cross edge : any other edge.

Properties of DFS : Assume we run DFS(G)

Parenthesis theorem: For any two vertices u, v

either : ① $[u.dtime, u.ftime) \cap [v.dtime, v.ftime) = \emptyset$

OR ② $[u.dtime, u.ftime) \subseteq [v.dtime, v.ftime)$

OR ③ $[u.dtime, u.ftime) \supseteq [v.dtime, v.ftime)$

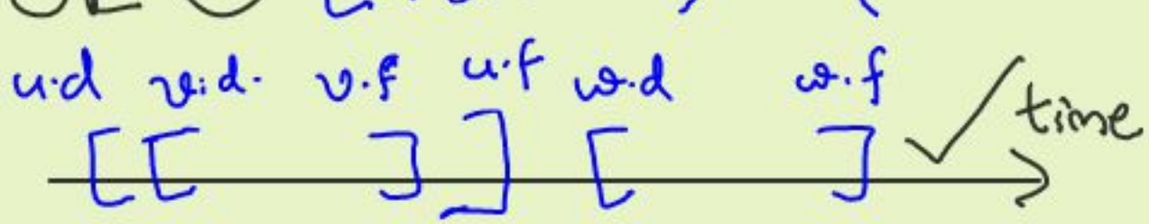
Properties of DFS : Assume we run DFS(G)

Parenthesis theorem: For any two vertices u, v

either : ① $[u.dtime, u.ftime) \cap [v.dtime, v.ftime) = \emptyset$

OR ② $[u.dtime, u.ftime) \subseteq [v.dtime, v.ftime)$

OR ③ $[u.dtime, u.ftime) \supseteq [v.dtime, v.ftime]$



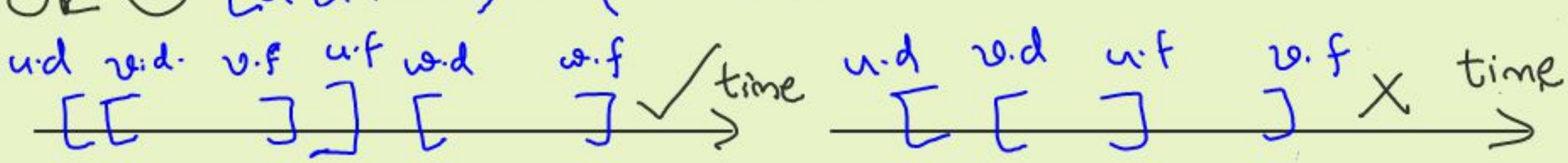
Properties of DFS : Assume we run DFS(G)

Parenthesis theorem: For any two vertices u, v

either : ① $[u.dtime, u.ftime) \cap [v.dtime, v.ftime) = \emptyset$

OR ② $[u.dtime, u.ftime) \subseteq [v.dtime, v.ftime)$

OR ③ $[u.dtime, u.ftime) \supseteq [v.dtime, v.ftime]$



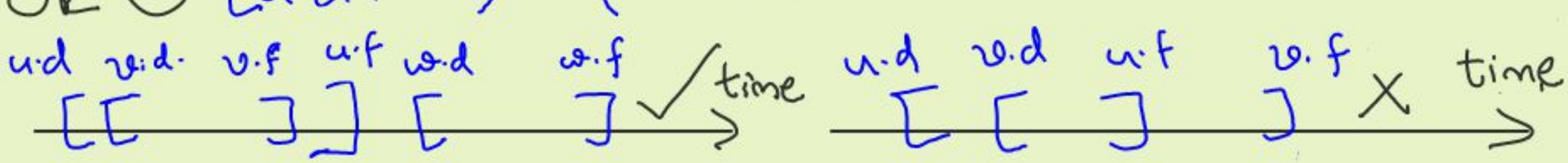
Properties of DFS : Assume we run DFS(G)

Parenthesis theorem: For any two vertices u, v

either : ① $[u.dtime, u.ftime) \cap [v.dtime, v.ftime) = \emptyset$

OR ② $[u.dtime, u.ftime) \subseteq [v.dtime, v.ftime)$

OR ③ $[u.dtime, u.ftime) \supseteq [v.dtime, v.ftime]$



Corollary: v is a descendant of u in DFS forest

⇕

③ holds, i.e. $u.dtime \leq v.dtime < v.ftime \leq u.ftime$

Properties of DFS : Assume we run DFS(G)

White-Path Theorem : v a descendant of u in DFS forest $\Longleftrightarrow$ at time $u.dtime$ there is a path from u to v consisting of only white vertices.

Properties of DFS : Assume we run DFS(G)

White-Path Theorem : v a descendant of u in DFS forest $\iff$ at time $u.dtime$ there is a path from u to v consisting of only white vertices.

Pf ($\Rightarrow$) If v a proper descendant of u, then $w.dtime > u.dtime$ forall w from u to v in DFS forest. This gives the white path.

Properties of DFS : Assume we run DFS(G)

White-Path Theorem : v a descendant of u in DFS forest $\iff$ at time $u.dtime$ there is a path from u to v consisting of only white vertices.

Pf ($\Rightarrow$) If v a proper descendant of u, then $w.dtime > u.dtime$ forall w from u to v in DFS forest. This gives the white path.

($\Leftarrow$) Induction on length l of white path.

Base: $l=0 \Rightarrow u=v$

Properties of DFS : Assume we run DFS(G)

White-Path Theorem : v a descendant of u in DFS forest $\iff$ at time $u.dtime$ there is a path from u to v consisting of only white vertices.

Pf ($\Rightarrow$) If v a proper descendant of u, then $w.dtime > u.dtime$ forall w from u to v in DFS forest. This gives the white path.

($\Leftarrow$) Induction on length l of white path.

Base: $l=0 \Rightarrow u=v$

Induction: Assume for $l \leq k$.

Properties of DFS : Assume we run DFS(G)

White-Path Theorem : v a descendant of u in DFS forest $\iff$ at time $u.dtime$ there is a path from u to v consisting of only white vertices.

Pf

Properties of DFS : Assume we run DFS(G)

White-Path Theorem : v a descendant of u in DFS forest $\iff$ at time $u.dtime$ there is a path from u to v consisting of only white vertices.

Pf

By Ind. hypothesis, w a descendant of u

Properties of DFS : Assume we run DFS(G)

White-Path Theorem : v a descendant of u in DFS forest $\iff$ at time $u.dtime$ there is a path from u to v consisting of only white vertices.

Pf

By Ind. hypothesis, w a descendant of u

By Parenthesis thm, $w.ftime \leq u.ftime$

Also $v.dtime < w.ftime$

Finally, $u.dtime < v.dtime$ (by W-path)

$\Rightarrow u.dtime < v.dtime < v.ftime \leq u.ftime$

Properties of DFS : Assume we run DFS(G)

White-Path Theorem : v a descendant of u in DFS forest $\iff$ at time $u.dtime$ there is a path from u to v consisting of only white vertices.

Pf

By Ind. hypothesis, w a descendant of u

By Parenthesis thm, $w.ftime \leq u.ftime$

Also $v.dtime < w.ftime$

Finally, $u.dtime < v.dtime$ (by W-path)

$\Rightarrow u.dtime < v.dtime < v.ftime \leq u.ftime$

$\Rightarrow v$ descendant of u

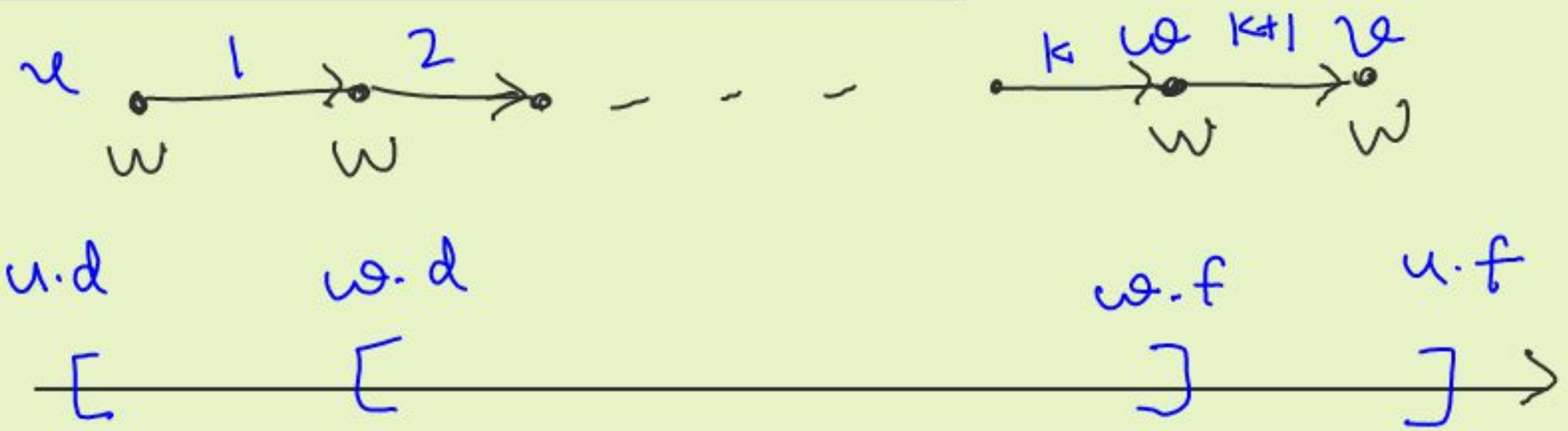
Picture for Inductive Case

By Induction Hypothesis, w a descendant of u &

by Parenthesis thm,

$$u.dtime \leq w.dtime < w.ftime \leq u.ftime$$

Picture for Inductive Case



When is v discovered?

→ Not before u : otherwise we don't have a W-path from u to v.

→ Not after v is finished : if v is white when processing v, it will be discovered.

Picture for Inductive Case

$\Longrightarrow$ v is discovered here

By Parenthesis theorem, v must finish before u.

$\Longrightarrow$ v is a descendant of u.