

Elementary
Graph Algorithms

PART-3

1

Outlines: Graphs Part-3

- Depth-First Search:
 - **DFS Algorithm**
 - **DFS Example**
 - **DFS Running Time**
 - **DFS Predecessor Subgraph**
 - **DFS Time Stamping**
 - **DFS Parenthesis Theorem**
 - **DFS Edge Classification**
 - **DFS and Graph Cycles**

2

Depth-First Search

- *Depth-first search* is another strategy for exploring a graph
 - Explore “deeper” in the graph whenever possible
 - Edges are explored out of the most recently discovered vertex v that still has unexplored edges
 - When all of v 's edges have been explored, **backtrack** to the vertex from which v was discovered

3

Depth-First Search

- Initialize
 - color all vertices white
- Visit each and every white vertex using DFS-Visit
- Each call to **DFS-Visit(u)** **roots a new tree** of the *depth-first forest* at vertex u
- A vertex is **white** if it is undiscovered
- A vertex is **gray** if it has been discovered but not all of its edges have been discovered
- A vertex is **black** after all of its adjacent vertices have been discovered (the adj. list was examined completely)

4

Depth-First Search

DFS(G)

```
1 for each vertex  $u \in V[G]$ 
2   do  $color[u] \leftarrow WHITE$ 
3  $time \leftarrow 0$ 
```

Init all
vertices

```
4 for each vertex  $u \in V[G]$ 
5   do if  $color[u] = WHITE$ 
6     then DFS-VISIT( $u$ )
```

DFS-VISIT(u)

```
1  $color[u] \leftarrow GRAY$            ▷ White vertex  $u$  discovered.
2  $d[u] \leftarrow time$              ▷ Mark with discovery time.
3  $time \leftarrow time + 1$          ▷ Tick global time.
```

```
4 for each  $v \in Adj[u]$              ▷ Explore all edges  $(u, v)$ .
5   do if  $color[v] = WHITE$ 
6     then DFS-VISIT( $v$ )
```

Visit all
children
recursively

```
7  $color[u] \leftarrow BLACK$        ▷ Blacken  $u$ ; it is finished.
8  $f[u] \leftarrow time$            ▷ Mark with finishing time.
9  $time \leftarrow time + 1$        ▷ Tick global time.
```

5

Depth-First Search: Algorithm

```
DFS( $G$ )
{
  for each vertex  $u \in V[G]$ 
  {
     $color[u] = WHITE$ ;
     $p[u] = NIL$ ;
  }
   $time = 0$ ;
  for each vertex  $u \in V[G]$ 
  {
    if ( $color[u] == WHITE$ )
      DFS_Visit( $u$ );
  }
}
```

```
DFS_Visit( $u$ )
{
   $color[u] = GRAY$ ;
   $time = time + 1$ ;
   $d[u] = time$ ;
  for each  $v \in Adj[u]$ 
    if ( $color[v] == WHITE$ )
       $p[v] = u$ ;
      DFS_Visit( $v$ );

   $color[u] = BLACK$ ;
   $time = time + 1$ ;
   $f[u] = time$ ;
}
```

6

Depth-First Search: Algorithm

```
DFS(G)
{
  for each vertex u ∈ V[G]
  {
    color[u] = WHITE;
    p[u] = NIL;
  }
  time = 0;
  for each vertex u ∈ V[G]
  {
    if (color[u] == WHITE)
      DFS_Visit(u);
  }
}

DFS_Visit(u)
{
  color[u] = GRAY;
  time = time+1;
  d[u] = time;
  for each v ∈ Adj[u]
  {
    if (color[v] == WHITE)
      p[v] = u;
      DFS_Visit(v);
  }
  color[u] = BLACK;
  time = time+1;
  f[u] = time;
}
```

What does $d[u]$ represent?

7

Depth-First Search: Algorithm

```
DFS(G)
{
  for each vertex u ∈ V[G]
  {
    color[u] = WHITE;
    p[u] = NIL;
  }
  time = 0;
  for each vertex u ∈ V[G]
  {
    if (color[u] == WHITE)
      DFS_Visit(u);
  }
}

DFS_Visit(u)
{
  color[u] = GRAY;
  time = time+1;
  d[u] = time;
  for each v ∈ Adj[u]
  {
    if (color[v] == WHITE)
      p[v] = u;
      DFS_Visit(v);
  }
  color[u] = BLACK;
  time = time+1;
  f[u] = time;
}
```

What does $f[u]$ represent?

8

Depth-First Search: Algorithm

```
DFS(G)                                DFS_Visit(u)
{
1  for each vertex u ∈ V[G]           1  color[u] = GRAY;
  {                                     2  time = time+1;
2    color[u] = WHITE;                3  d[u] = time;
3    p[u] = NIL                        4  for each v ∈ Adj[u]
  }                                     5    if (color[v] == WHITE)
4  time = 0;                           6    p[v] = u;
5  for each vertex u ∈ V[G]           7    DFS_Visit(v);
  {                                     8  color[u] = BLACK;
6    if (color[u] == WHITE)           9  time = time+1;
7    DFS_Visit(u);                   10 f[u] = time;
  }
}
```

Will all vertices eventually be colored black?

9

Depth-First Search: DFS Procedure

- Lines 1-3 paint all vertices white and initialize their p fields to NIL.
- Line 4 resets the global time counter.
- Lines 5-7 check each vertex in V in turn and, when a white vertex is found, visit it using DFS_Visit.
 - Every time DFS_Visit(u) is called in line 7, vertex u becomes the root of a new tree in the depth-first forest.
- When DFS returns, every vertex u has been assigned a *discovery time* d[u] and a *finishing time* f[u].

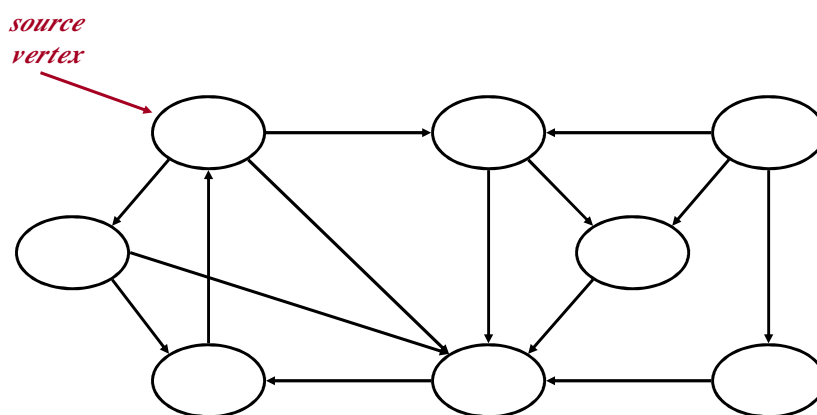
10

Depth-First Search: DFS-Visit Procedure

- In each call $\text{DFS-Visit}(u)$, vertex u is initially white.
- Line 1 paints u gray.
- Line 2 increments the global variable time .
- Line 3 records the new value of time as the discovery time $d[u]$.
- Lines 4-7 examine each vertex v adjacent to u and recursively visit v if it is white.
 - As each vertex $v \in \text{Adj}[u]$ is considered in line 4, we say that edge (u, v) is *explored* by the depth-first search.
- Finally, after every edge leaving u has been explored, lines 8-10 paint u black and record the finishing time in $f[u]$.

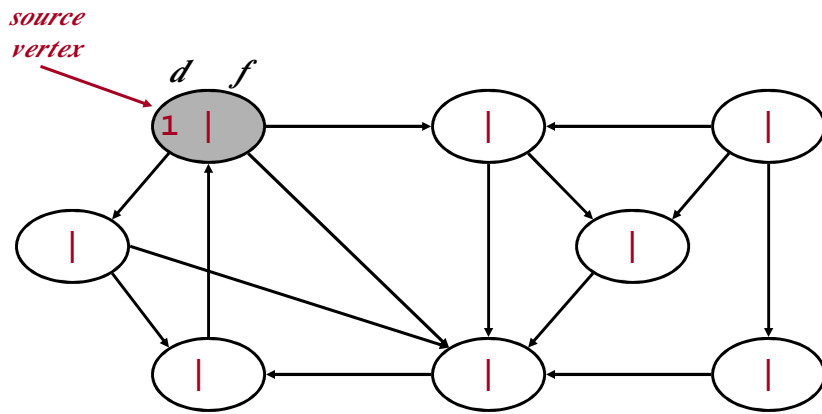
11

DFS Example



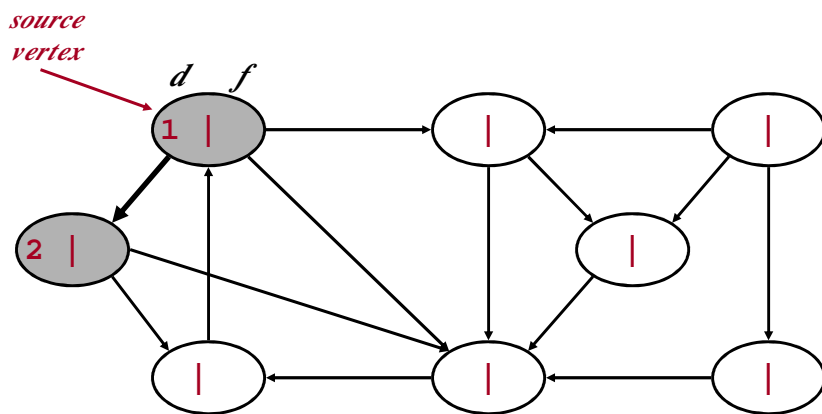
12

DFS Example



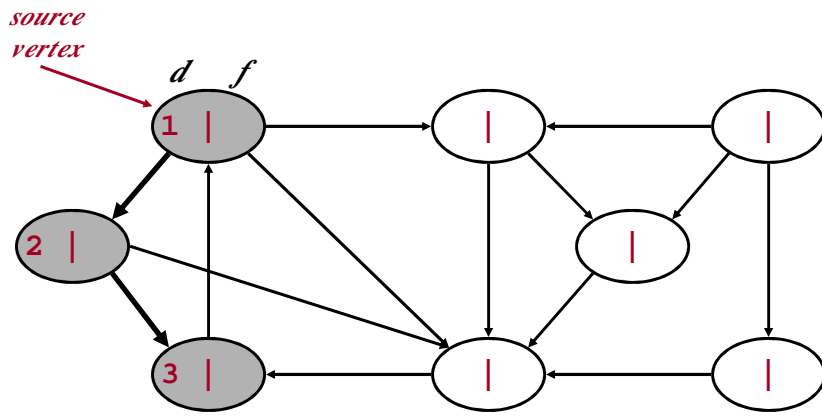
13

DFS Example



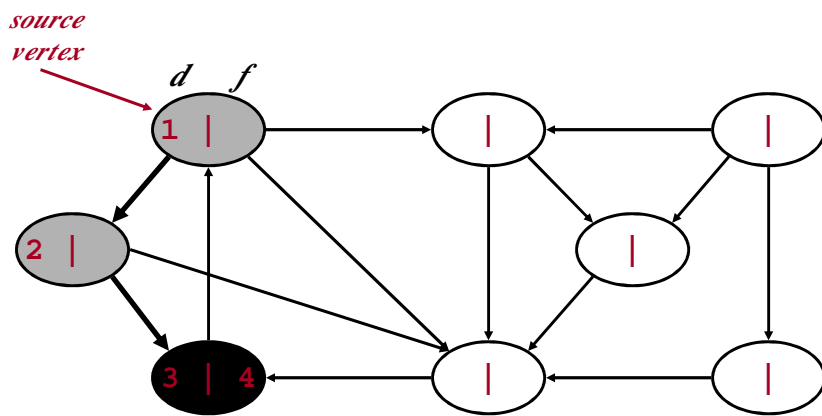
14

DFS Example



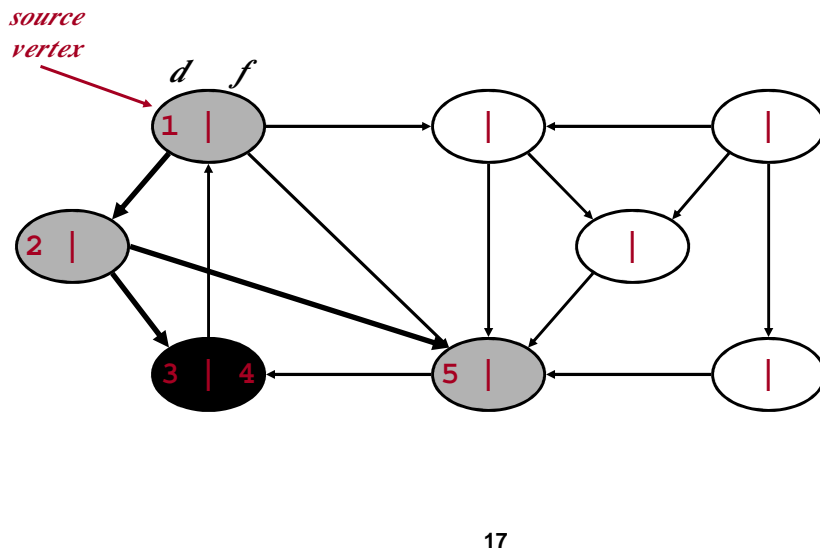
15

DFS Example

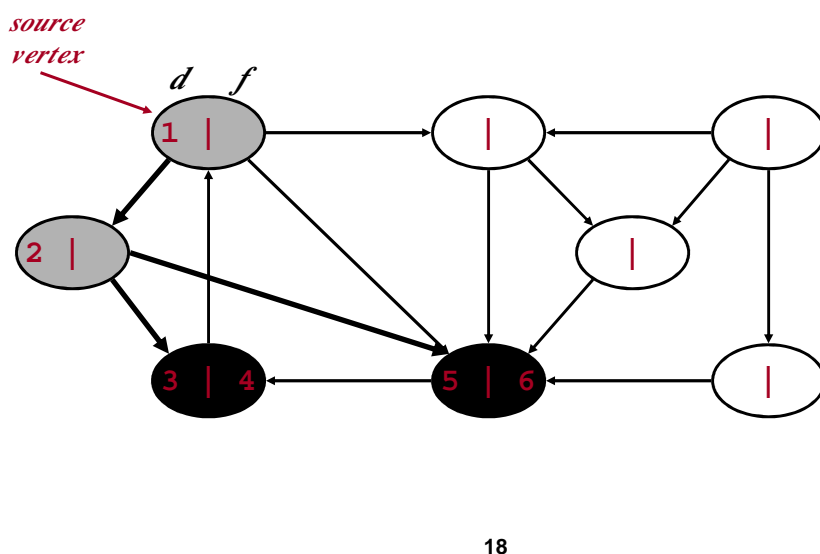


16

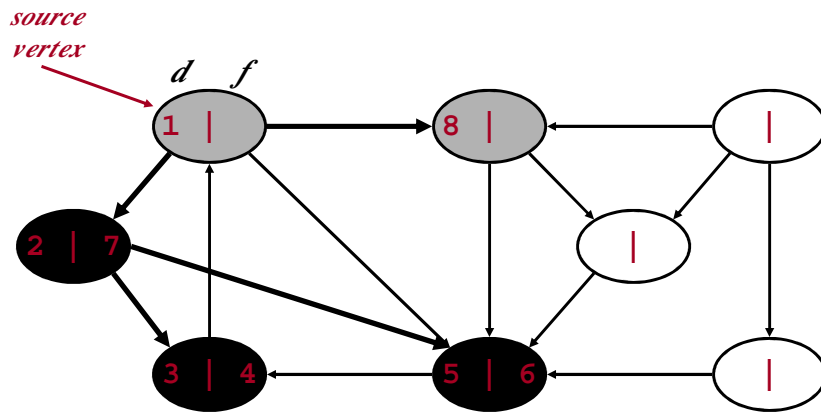
DFS Example



DFS Example

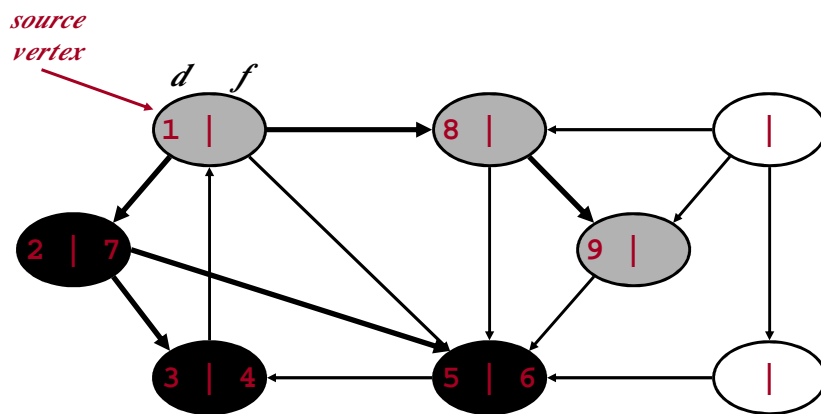


DFS Example



19

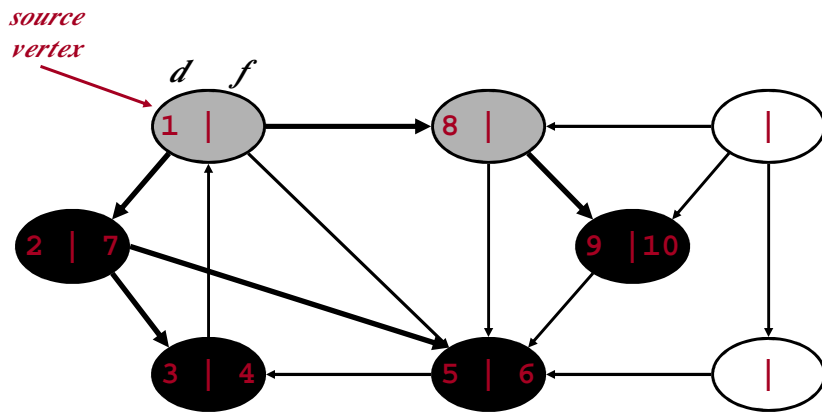
DFS Example



*What is the structure of the gray vertices?
What do they represent?*

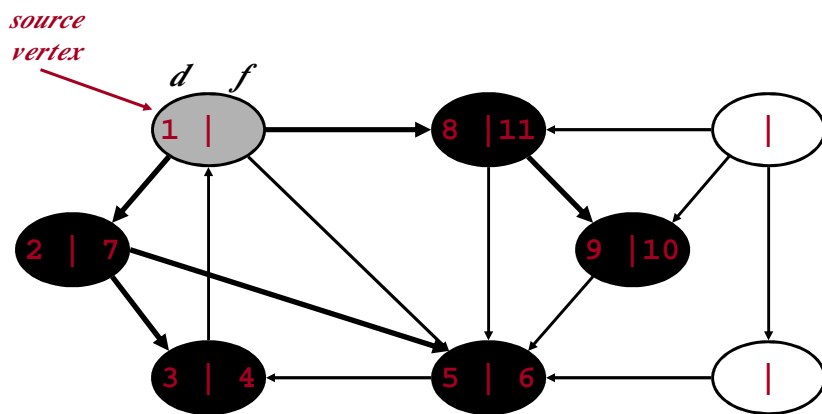
20

DFS Example



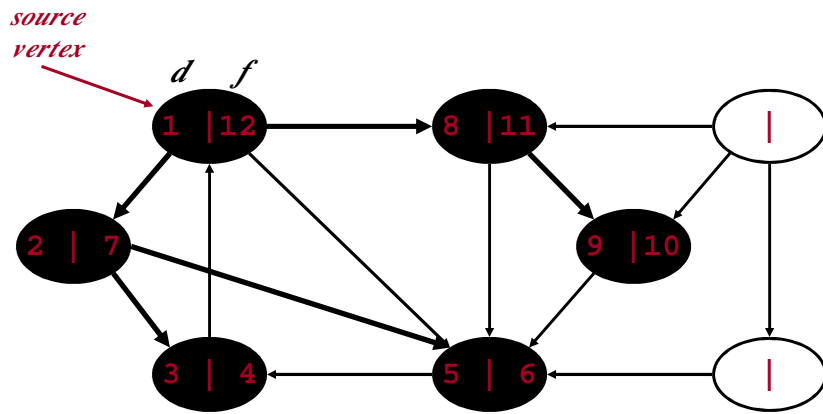
21

DFS Example



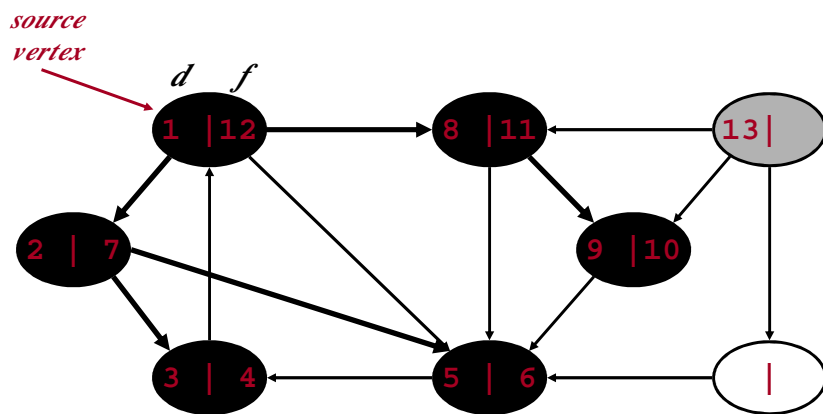
22

DFS Example



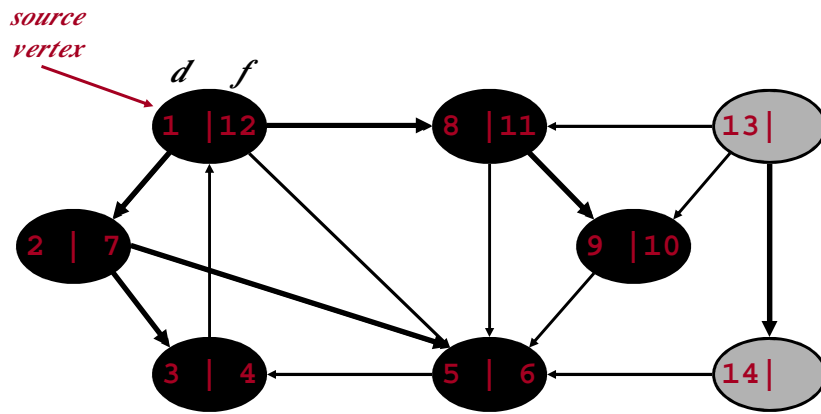
23

DFS Example



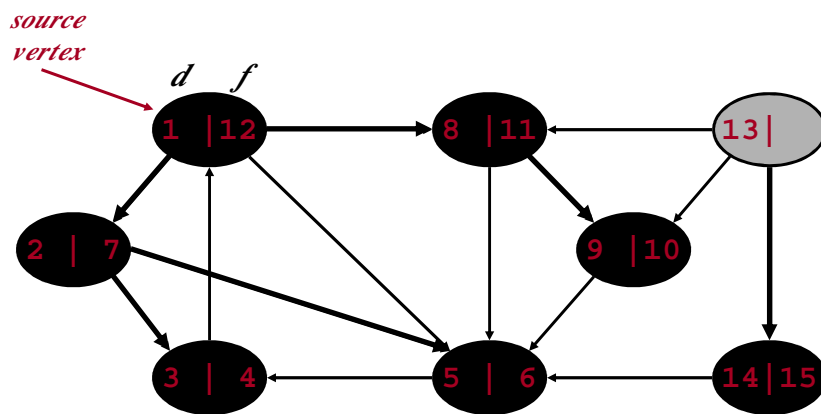
24

DFS Example



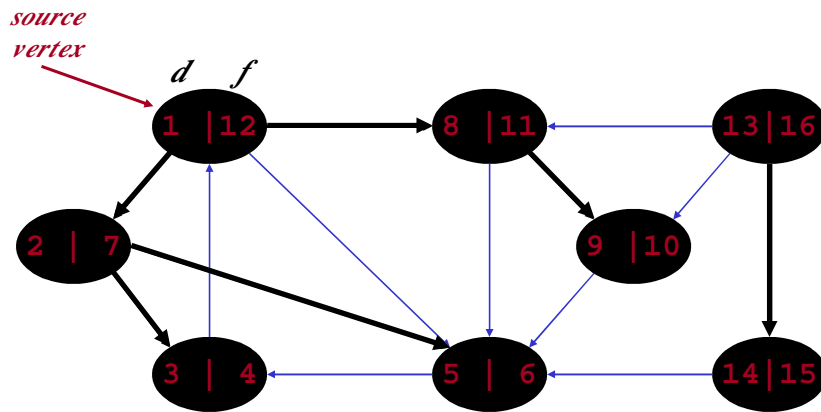
25

DFS Example



26

DFS Example



27

Depth-First Search: Algorithm

```

DFS(G)
{
  for each vertex u ∈ V[G]
  {
    color[u] = WHITE;
    p[u] = NIL
  }
  time = 0;
  for each vertex u ∈ V[G]
  {
    if (color[u] == WHITE)
      DFS_Visit(u);
  }
}

DFS_Visit(u)
{
  color[u] = GRAY;
  time = time+1;
  d[u] = time;
  for each v ∈ Adj[u]
  {
    if (color[v] == WHITE)
      p[v] = u;
      DFS_Visit(v);
  }
  color[u] = BLACK;
  time = time+1;
  f[u] = time;
}

```

What will be the running time?

28

Depth-First Search: Algorithm

```

DFS(G)
{
  for each vertex u ∈ V[G]
  {
    color[u] = WHITE;
    p[u] = NIL
  }
  time = 0;
  for each vertex u ∈ V[G]
  {
    if (color[u] == WHITE)
      DFS_Visit(u);
  }
}

DFS_Visit(u)
{
  color[u] = GRAY;
  time = time+1;
  d[u] = time;
  for each v ∈ Adj[u]
  {
    if (color[v] == WHITE)
      p[v] = u;
      DFS_Visit(v);
  }
  color[u] = BLACK;
  time = time+1;
  f[u] = time;
}

```

How many times will DFS_Visit() actually be called? $\Theta(V)$
 How many times will *if* statement be executed in all $\Theta(V)$ DFS_Visit calls?
 $\Theta(E)$

29

Depth-First Search: Algorithm

```

DFS(G)
{
  for each vertex u ∈ V[G]
  {
    color[u] = WHITE;
    p[u] = NIL
  }
  time = 0;
  for each vertex u ∈ V[G]
  {
    if (color[u] == WHITE)
      DFS_Visit(u);
  }
}

DFS_Visit(u)
{
  color[u] = GRAY;
  time = time+1;
  d[u] = time;
  for each v ∈ Adj[u]
  {
    if (color[v] == WHITE)
      p[v] = u;
      DFS_Visit(v);
  }
  color[u] = BLACK;
  time = time+1;
  f[u] = time;
}

```

So, running time of DFS = $\Theta(V+E)$

30

DFS: Running Time

- Running time
 - the loops in **DFS** take time $\Theta(V)$ each, *excluding* the time to execute **DFS-Visit**
 - **DFS-Visit** is called once for every vertex
 - its only invoked on white vertices, and
 - paints the vertex gray immediately
 - for each **DFS-Visit**(v) a loop iterates over all $\text{Adj}[v]$
 - the *total cost* for **DFS-Visit** is $\Theta(E)$

$$\sum_{v \in V} |\text{Adj}[v]| = \Theta(E)$$

- the running time of DFS is $\Theta(V+E)$

31

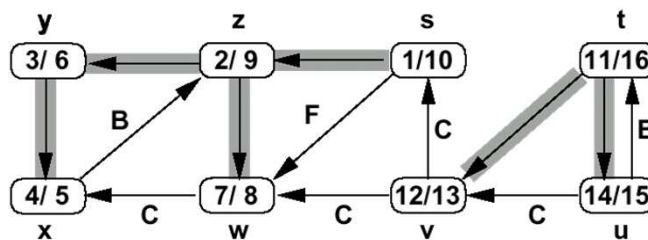
Predecessor Subgraph

- Define slightly different from **BFS**
- Defined the *predecessor subgraph* of G as $G_p = (V, E_p)$, where
 - $E_p = \{(p[v], v) \in E : v \in V \text{ and } p[v] \neq \text{NIL}\}$
- The PD subgraph of a depth-first search forms a **depth-first forest** composed of several depth-first trees
- The edges in G_p are called *tree edges*

32

DFS Timestamping

- The **DFS** algorithm maintains a monotonically increasing global clock
 - discovery time $d[u]$ and finishing time $f[u]$
- For every vertex u , the inequality $d[u] < f[u]$ must hold



33

DFS Timestamping

- Vertex u is
 - white before time $d[u]$
 - gray between time $d[u]$ and time $f[u]$, and
 - black thereafter
- Notice the structure throughout the algorithm.
 - gray vertices form a *linear chain*
 - corresponds to a *stack* of vertices that have not been exhaustively explored (DFS-Visit started but not yet finished)
 - What is the *size* of the *stack* in DFS?

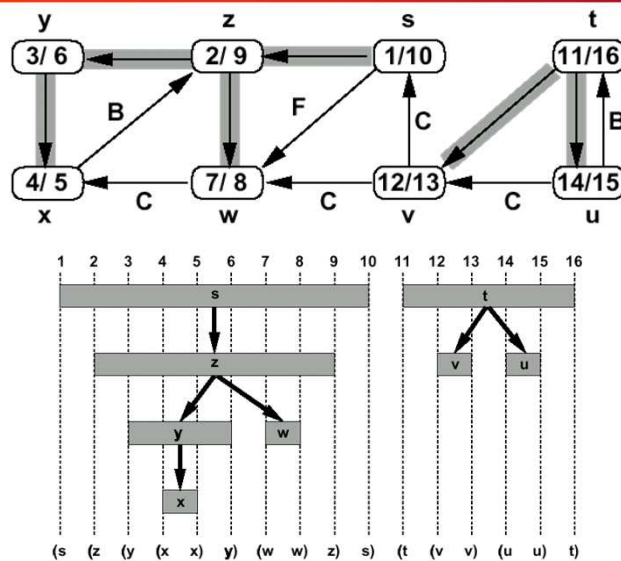
34

DFS Parenthesis Theorem

- Discovery and finish times have parenthesis structure
 - represent *discovery* of u with *left* parenthesis " $(u$ "
 - represent *finishing* of u with *right* parenthesis " $)u$ "
 - history of discoveries and finishings makes a well-formed expression (parenthesis are properly nested)

35

DFS Parenthesis Theorem



36

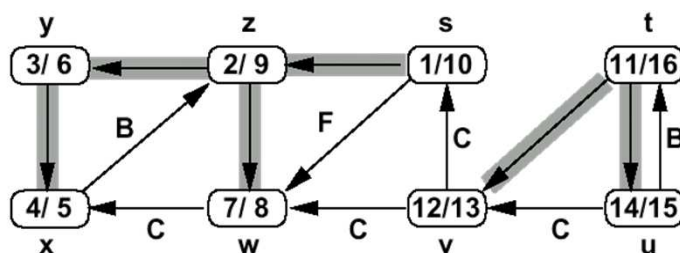
DFS Parenthesis Theorem

- For any two distinct vertices u and v , exactly one of the following **three conditions** holds:
 - The intervals $[d[u], f[u]]$ and $[d[v], f[v]]$ are *disjoint*
 - neither u nor v is a **descendant** (child) of the other in the depth-first forest
 - The interval $[d[u], f[u]]$ is contained entirely within the interval $[d[v], f[v]]$
 - $d[v] < d[u] < f[u] < f[v]$
 - u is a **descendant** of v in the depth-first forest
 - The interval $[d[v], f[v]]$ is contained entirely within the interval $[d[u], f[u]]$
 - $d[u] < d[v] < f[v] < f[u]$
 - v is a **descendant** of u in the depth-first forest

37

DFS Edge Classification

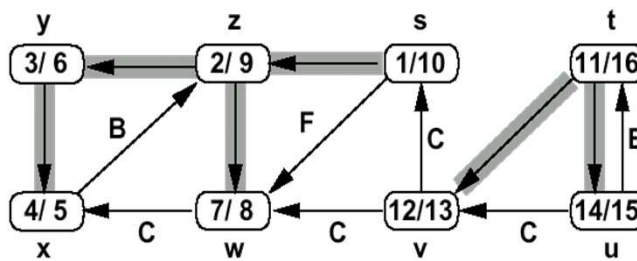
- **Tree edge** (gray to white)
 - encounter new vertices (white)
 - included in **depth-first forest** G_p
- **Back edge** (gray to gray)
 - from *descendant* to *ancestor*
 - non-tree edges in **depth-first tree**



38

DFS Edge Classification

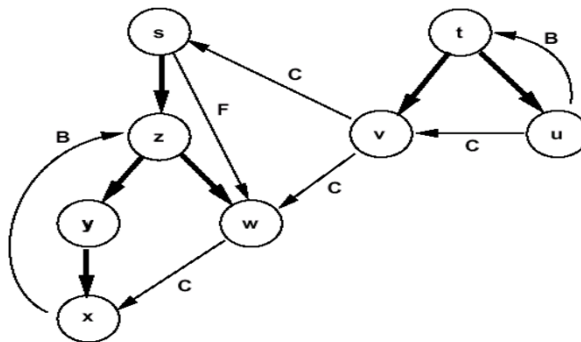
- **Forward edge** (gray to black): if $d[u] < d[v]$
 - from *ancestor to descendant*
 - non-tree edges in **depth-first tree**
- **Cross edge** (gray to black): if $d[u] > d[v]$
 - are all other edges
 - between trees or subtrees



39

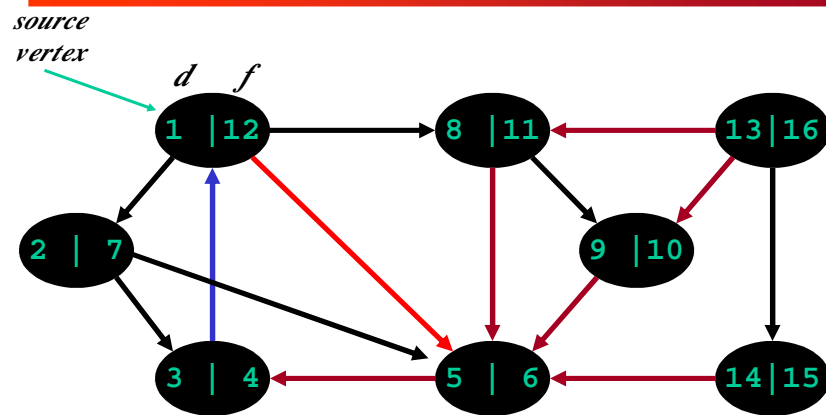
DFS Edge Classification

- **Tree** and **back** edges are important
- Most algorithms do not distinguish between **forward** and **cross** edges



40

DFS Example



Tree edges Back edges Forward edges Cross edges

41

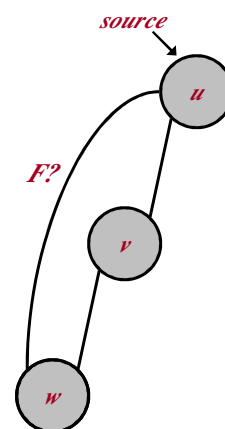
DFS: Kinds Of Edges

○ Theorem 22.10:

- If G is **undirected**, a DFS produces only **tree** and **back** edges

○ Proof by contradiction:

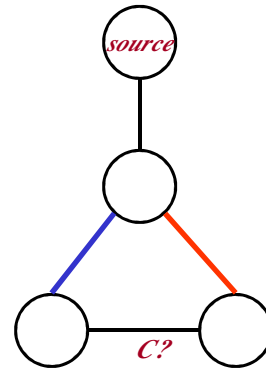
- Assume there's a **forward** edge
 - But $F?$ edge must actually be a **back** edge (*why?*)
 - Since edge (w, u) is from *descendant to ancestor* (*gray to gray*)



42

DFS: Kinds Of Edges

- **Theorem 22.10:**
 - If G is **undirected**, a DFS produces only **tree** and **back** edges
- Proof by contradiction:
 - Assume there's a **cross** edge
 - But $C?$ edge **cannot** be **cross**:
 - must be explored from one of the vertices it connects, becoming a **tree** vertex, before other vertex is explored
 - So in fact the picture is wrong...**both blue** and **red** edges **cannot** in fact be **tree** edges.
 - One should be **tree** edge and
 - the other should be **back** edge (cycle graph)



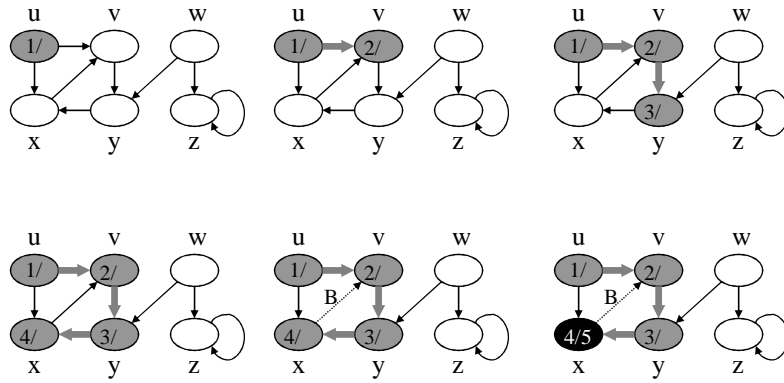
43

DFS and Graph Cycles

- Theorem:
- An **undirected** graph is *acyclic* (no cycles) iff a DFS yields **no back** edges
 - If acyclic, no back edges (because a back edge implies a cycle)
 - If no back edges, acyclic
 - No back edges implies only tree edges (*Why?*)
 - Only *tree edges* implies we have a *tree or a forest*
 - Which by definition is acyclic
- Thus, can run DFS to find whether a graph has a cycle. (*How*)

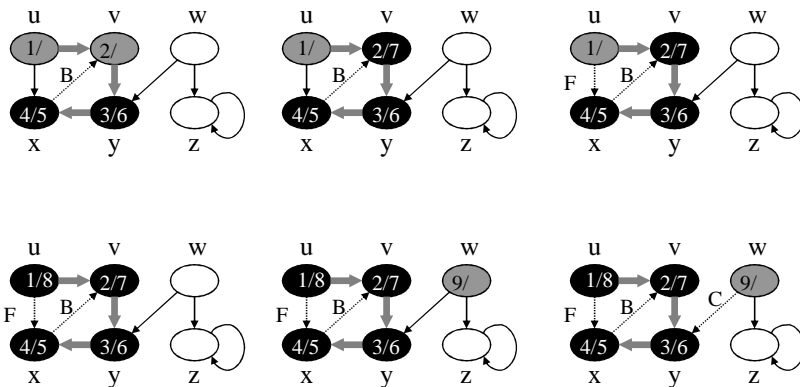
44

DFS Example



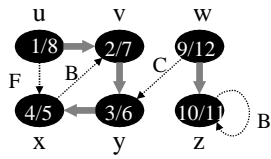
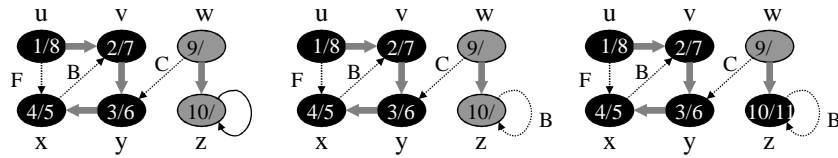
45

DFS Example



46

DFS Example



If you have a back edge in a directed graph, then you may have a cycle or must have a cycle?