Graphs

Lecture 18

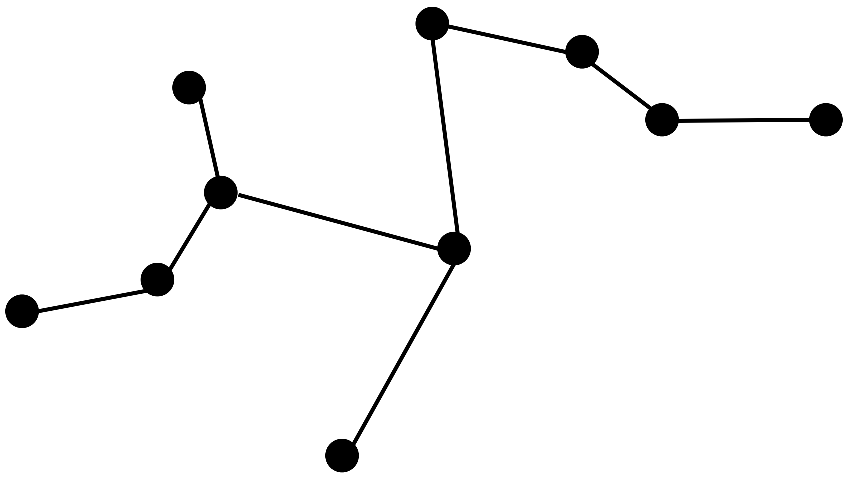
Instructor: Dr. Cong Pu, Ph.D.

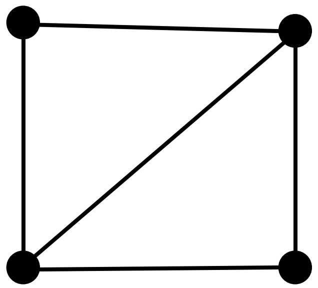
cong.pu@okstate.edu

Adapted partially from Data Structures and Algorithms in Java, M.T. Goodrich, R.Tamassia and M. H. Goldwasser, Sixth Edition, Wiley; Data Structures and Algorithms in C++, Adam Drozdek, 4th Edition, Cengage Learning

Acyclic Graph

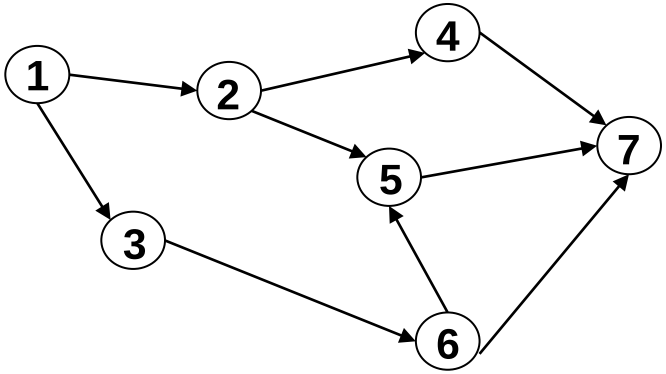
- An acyclic graph is a graph without cycles
  - A cycle is a complete circuit





Directed Acyclic Graph

- A directed acyclic graph (DAG) is an acyclic graph that has a direction

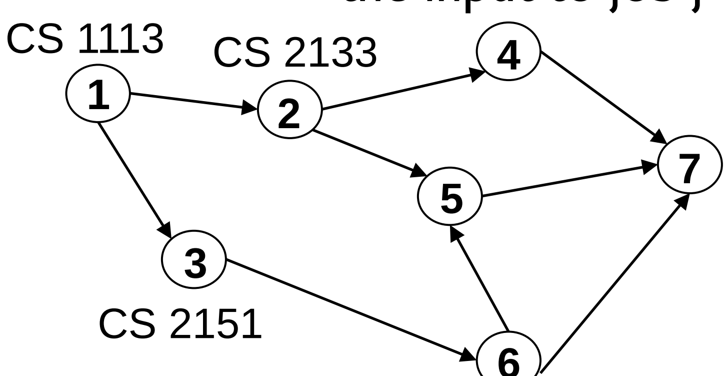


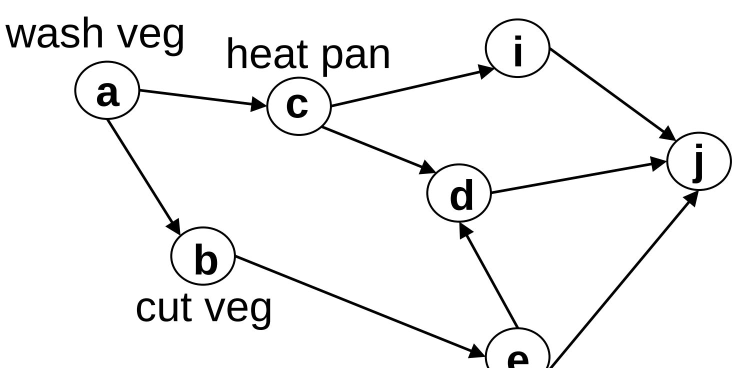
Vertex Set = {1, 2, 3, 4, 5, 6, 7}

Edge Set = {{1, 2}, {1, 3}, {2, 4}, {2. 5}, {3, 6}, {4, 7}, {5, 7}, {6, 7}, {6, 5}}

Directed Acyclic Graph (cont.)

- DAGs are a very common structure in computer science
  - many kinds of dependency networks
- DAGs can be used to encode ***precedence relations*** or ***dependencies*** in a natural way
  - for example
    - the vertex may be courses, with prerequisite requirements
    - the vertex may correspond to a pipeline of computing jobs, with assertions that the output of job i is used in determining the input to job j

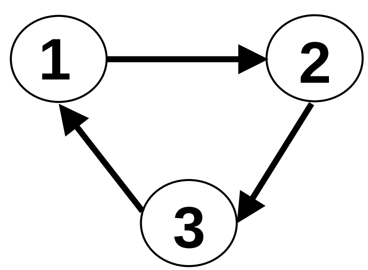




Directed Acyclic Graph (cont.)

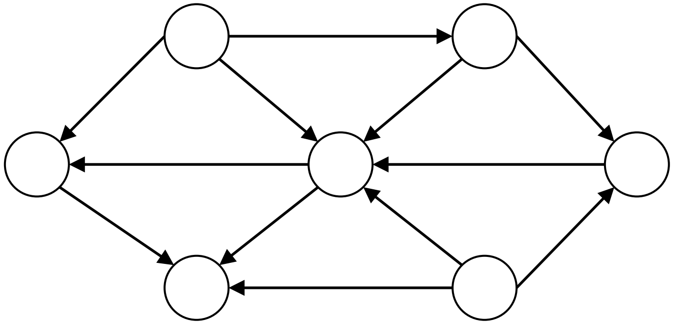
- Represent such an interdependent set of tasks
  - introduce a vertex for each task
  - introduce a directed edge (i, j) whenever i must be done before j

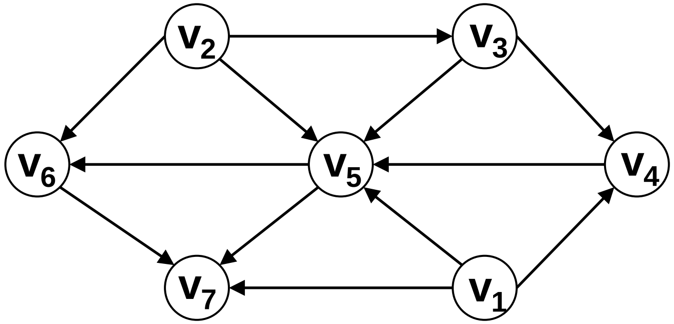
- If the precedence relation is to be at all meaningful, the resulting graph G must be DAG
  - if containing a cycle C, there would be no way to do any of the tasks in C
    - since each task in C cannot begin until some other one completes
    - no task in C could ever be done, since none could be done first



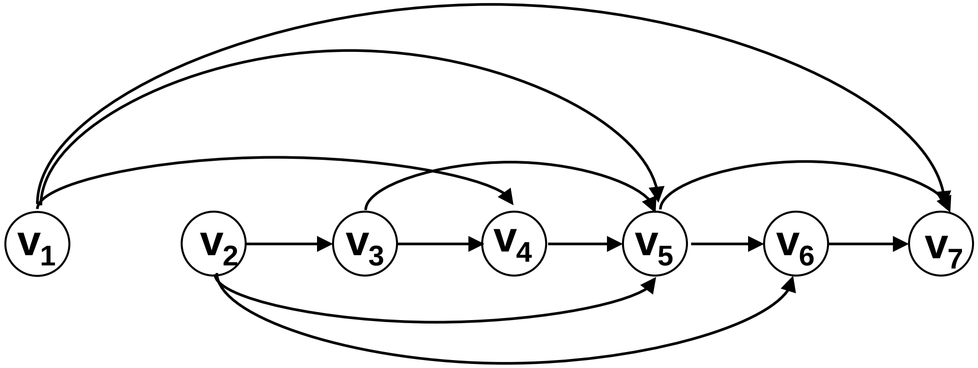
Directed Acyclic Graph (cont.)

- Directed acyclic graph





- The same graph with vertex ordering



Topological Ordering

- For a directed graph G, a *topological ordering* of G is an ordering of its nodes as $v_1, v_2, \ldots, v_n$, so that for every edge (v_i, v_j), we have $i < j$
  - in other words, all edges point "forward" in the ordering

- A *topological ordering* on tasks provides an order in which they can be safely performed
  - when we come to the task v_j, all the tasks that are required to precede it have already been done

Design Algorithm

- Add color: color vertices during the search to indicate their state
  - each vertex is initially ***white***
  - each vertex is colored ***gray*** when it is discovered in the search
  - each vertex is colored ***black*** when its *adjacency list* has been examined completely

- Add timestamp
  - each vertex v has two timestamps:
    - the first timestamp ***v.d*** records when ***v*** is first discovered (***grayed***)
    - the second timestamp ***v.f*** records when the search finishes examining ***v***'s adjacency list (***blacked***)

New Version of DFS

- Pseudocode

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL    // u.π : predecessor of u
time = 0         // timestamp (global)
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

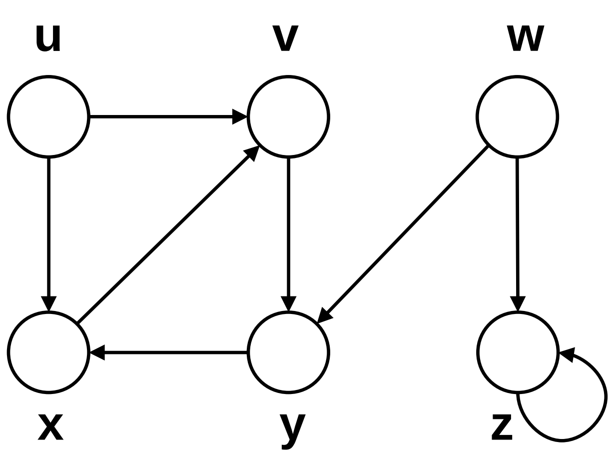
```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)    // recursion
u.color = BLACK
time = time + 1
u.f = time
```

Note:

This version of DFS is using ***recursion*** to keep track of searching, ***not*** the Stack.

New Version of DFS (co

time = 0

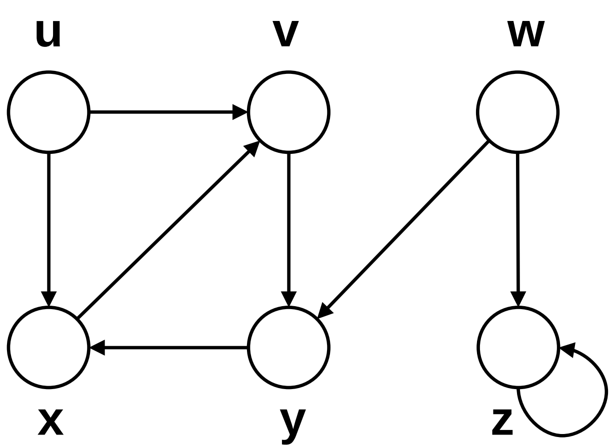



```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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
```

New Version of DFS (co

time = 0

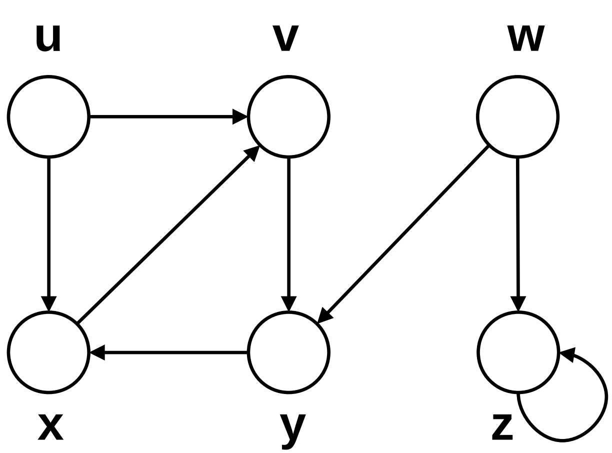



```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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
```

New Version of DFS (co…

time = 0




```
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
```

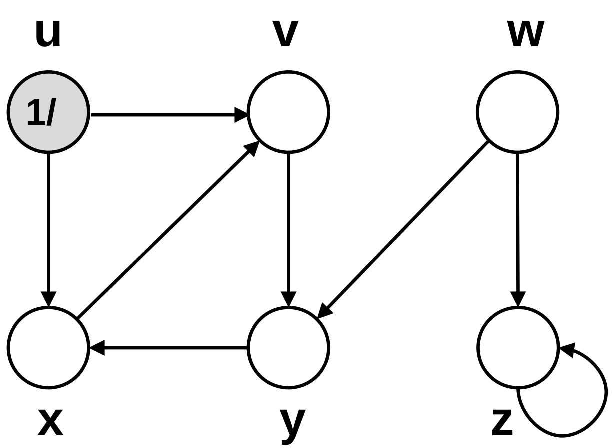
New Version of DFS (co

```
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
```

time = 1

u.d: discovery time
u.f: finish time
u.d / u.f

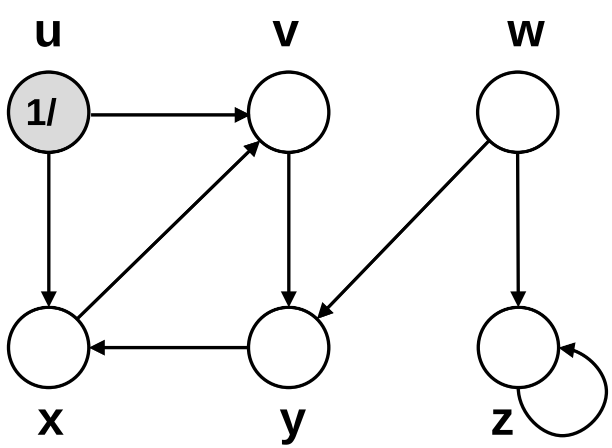


New Version of DFS (con

```
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)
```

time = 1

u.d: discovery time
u.f: finish time
u.d / u.f

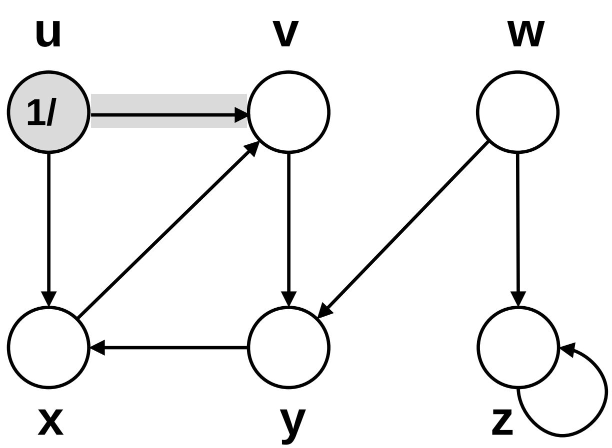



```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

New Version of DFS (co

time = 1

u.d: discovery time
u.f: finish time
u.d / u.f



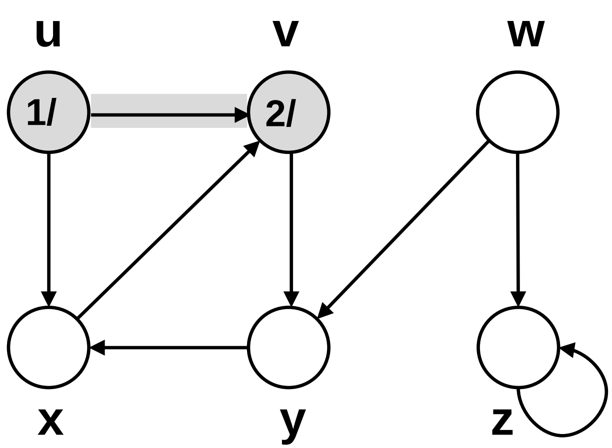

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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) // recursion
u.color = BLACK
time = time + 1
u.f = time
```

New Version of DFS (co

time = 2

u.d: discovery time
u.f: finish time
u.d / u.f



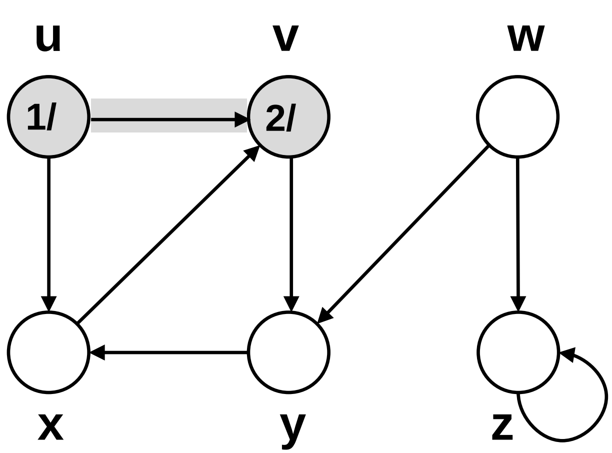

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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
```

New Version of DFS (cor

time = 2

u.d: discovery time
u.f: finish time
u.d / u.f



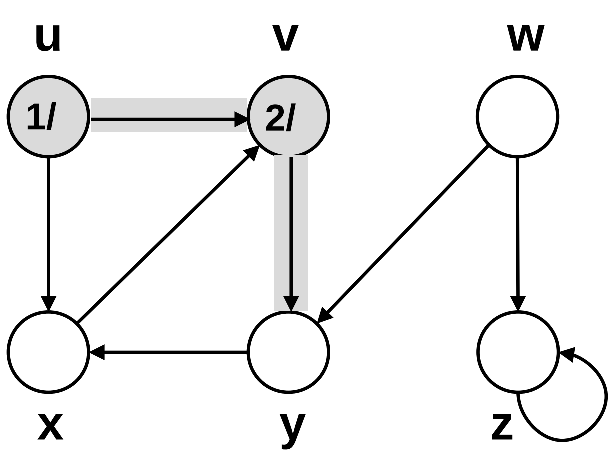

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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
```

New Version of DFS (co

time = 2

u.d: discovery time
u.f: finish time
u.d / u.f



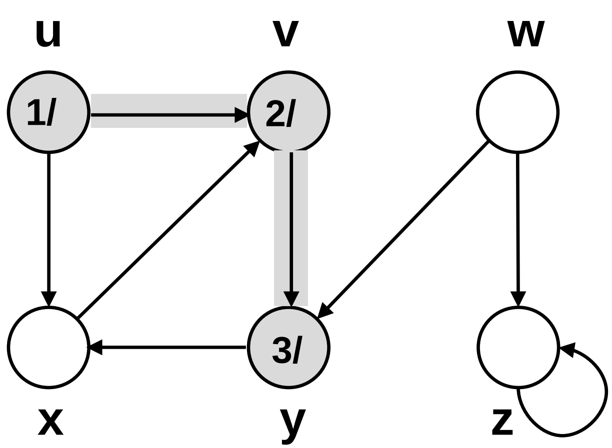

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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) // recursion
u.color = BLACK
time = time + 1
u.f = time
```

New Version of DFS (cor

time = 3

u.d: discovery time
u.f: finish time
u.d / u.f



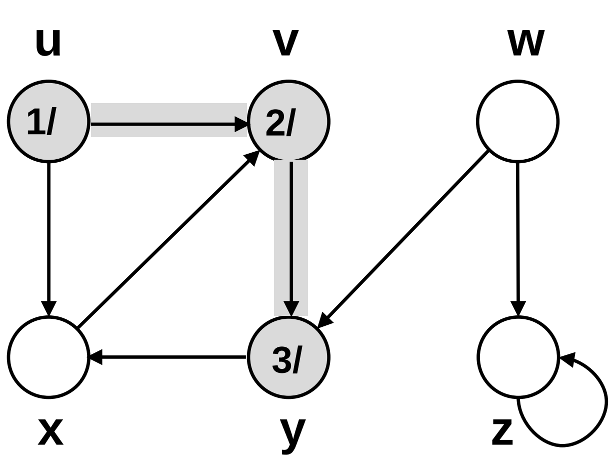

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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
```

New Version of DFS (co

time = 3

u.d: discovery time
u.f: finish time
u.d / u.f



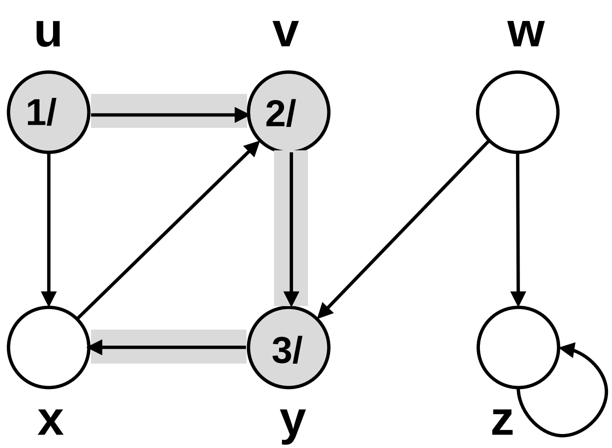

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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
```

New Version of DFS (cor

time = 3

u.d: discovery time
u.f: finish time
u.d / u.f



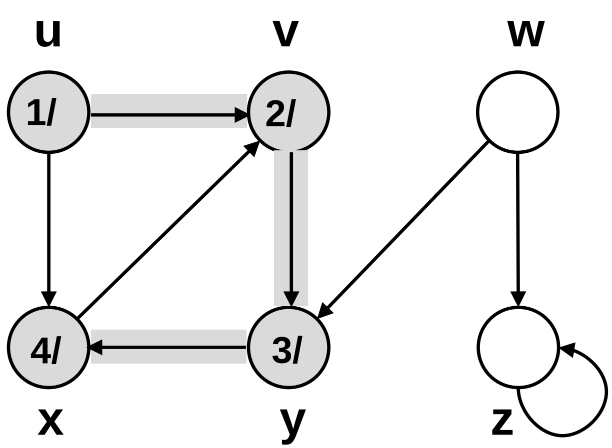
DFS(G)
1. for each vertex u ∈ G.V
2.   u.color = WHITE
3.   u.π = NIL
4. time = 0
5. for each vertex u ∈ G.V
6.   if u.color == WHITE
7.   DFS-VISIT(G, u)

DFS-VISIT(G, u)
1. time = time + 1
2. u.d = time
3. u.color = GRAY
4. **for each v ∈ G.Adj[u]**
5. **if v.color == WHITE**
6. **v.π = u**
7. **DFS-VISIT(G, v) // recursion**
8. u.color = BLACK
9. time = time + 1
10. u.f = time

New Version of DFS (cor

time = 4

u.d: discovery time
u.f: finish time
u.d / u.f



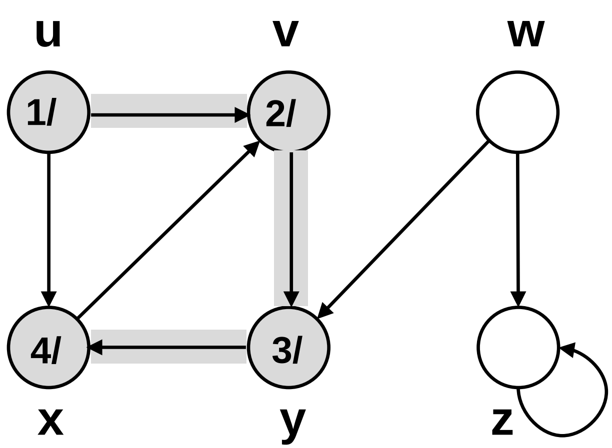

```
DFS(G)
1. for each vertex u ∈ G.V
2.     u.color = WHITE
3.     u.π = NIL
4.  time = 0
5.  for each vertex u ∈ G.V
6.      if u.color == WHITE
7.          DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
1.  time = time + 1
2.  u.d = time
3.  u.color = GRAY
4.  for each v ∈ G.Adj[u]
5.      if v.color == WHITE
6.          v.π = u
7.          DFS-VISIT(G, v)
8.   u.color = BLACK
9.   time = time + 1
10.  u.f = time
```

New Version of DFS (co

time = 4

u.d: discovery time
u.f: finish time
u.d / u.f




```
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)
```

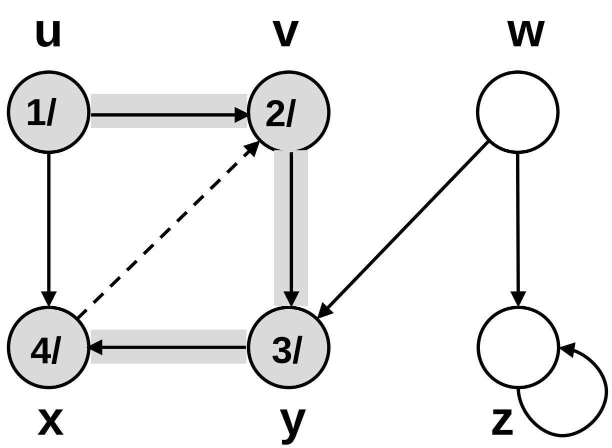
```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

New Version of DFS (co

```
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)
```

time = 4

u.d: discovery time
u.f: finish time
u.d / u.f

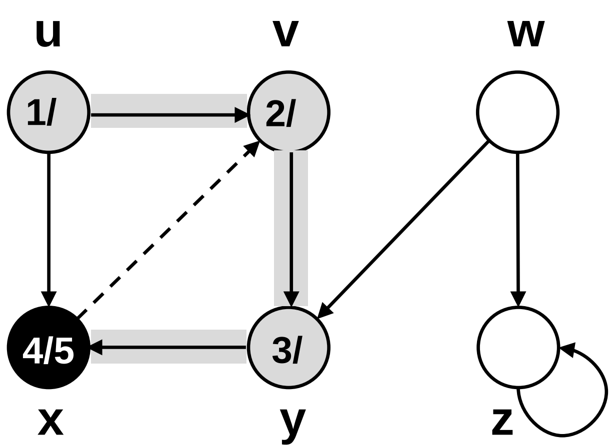



```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

New Version of DFS (co[ntinued]

time = 5

u.d: discovery time
u.f: finish time
u.d / u.f



DFS(G)
1. for each vertex u ∈ G.V
2. u.color = WHITE
3. u.π = NIL
4. time = 0
5. for each vertex u ∈ G.V
6. if u.color == WHITE
7. DFS-VISIT(G, u)

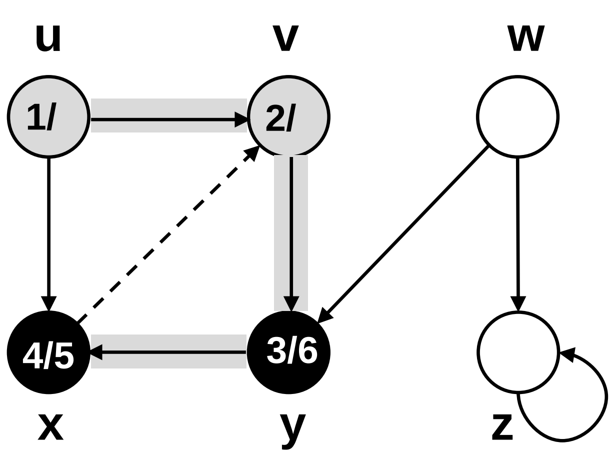
DFS-VISIT(G, u)
1. time = time + 1
2. u.d = time
3. u.color = GRAY
4. for each v ∈ G.Adj[u]
5. if v.color == WHITE
6. v.π = u
7. DFS-VISIT(G, v)
8. **u.color = BLACK**
9. **time = time + 1**
10. **u.f = time**

New Version of DFS (con

```
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)
```

time = 6

u.d: discovery time
u.f: finish time
u.d / u.f

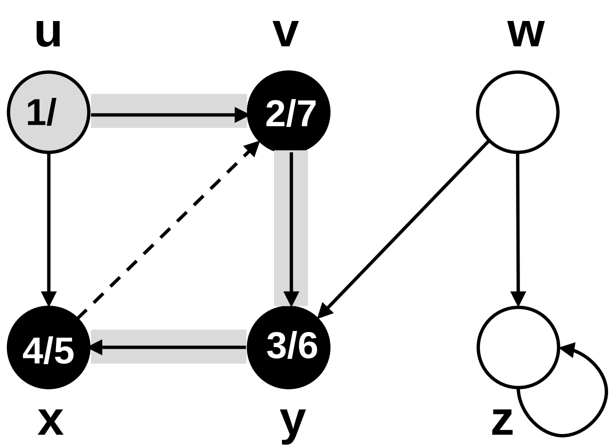



```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

New Version of DFS (co

time = 7

u.d: discovery time
u.f: finish time
u.d / u.f



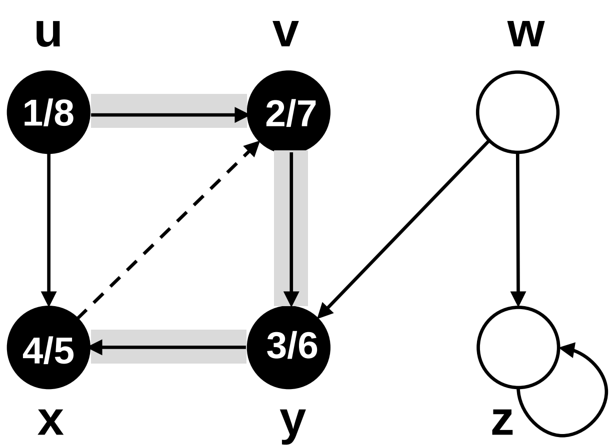

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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
```

New Version of DFS (co

time = 8

u.d: discovery time
u.f: finish time
u.d / u.f



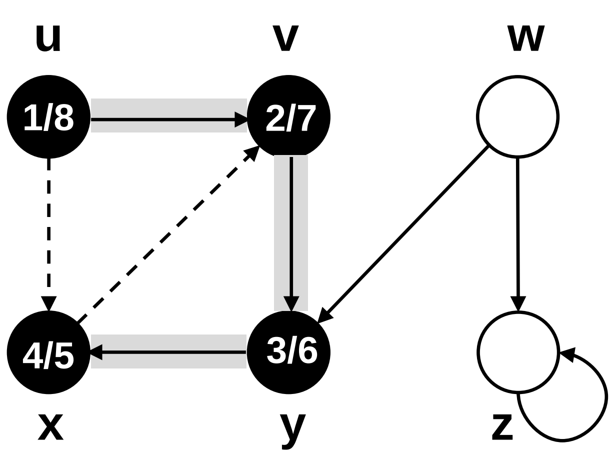

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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
```

New Version of DFS (co

time = 8

u.d: discovery time
u.f: finish time
u.d / u.f



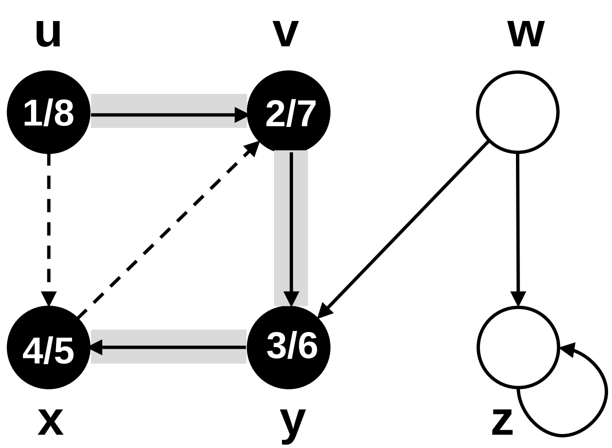

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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
```

New Version of DFS (con

time = 8

u.d: discovery time
u.f: finish time
u.d / u.f



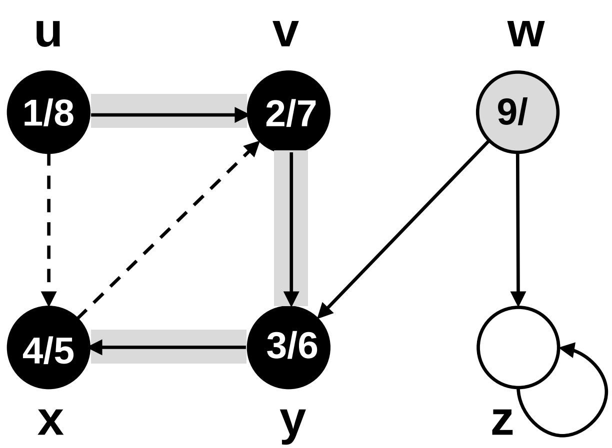
DFS(G)
1. for each vertex u ∈ G.V
2. u.color = WHITE
3. u.π = NIL
4. time = 0
5. **for each vertex u ∈ G.V**
6. **if u.color == WHITE**
7. **DFS-VISIT(G, u)**

DFS-VISIT(G, u)
1. time = time + 1
2. u.d = time
3. u.color = GRAY
4. for each v ∈ G.Adj[u]
5. if v.color == WHITE
6. v.π = u
7. DFS-VISIT(G, v)
8. u.color = BLACK
9. time = time + 1
10. u.f = time

New Version of DFS (con

time = 9

u.d: discovery time
u.f: finish time
u.d / u.f




```
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)
```

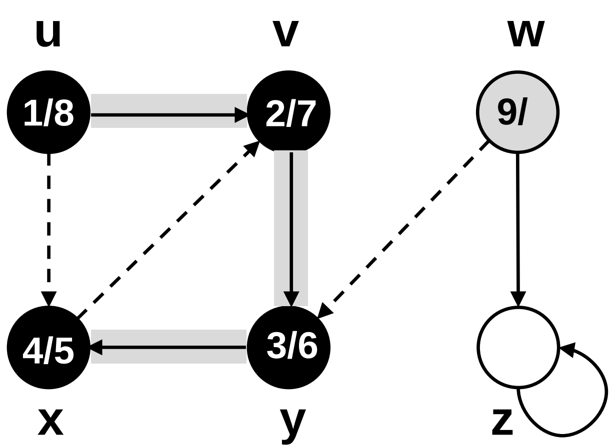
```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

New Version of DFS (co

DFS(G)
1. for each vertex u ∈ G.V
2. u.color = WHITE
3. u.π = NIL
4. time = 0
5. for each vertex u ∈ G.V
6. if u.color == WHITE
7. DFS-VISIT(G, u)

time = 9

u.d: discovery time
u.f: finish time
u.d / u.f

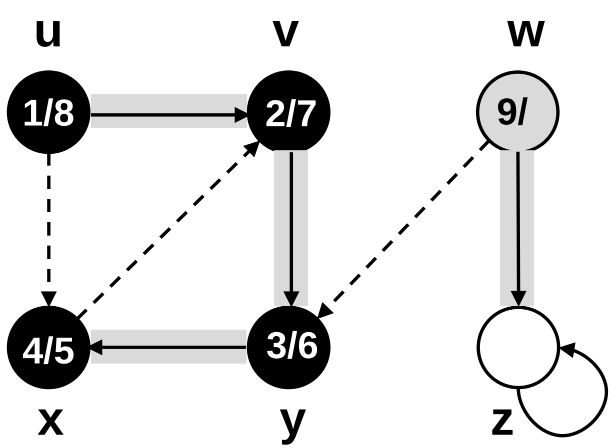


DFS-VISIT(G, u)
1. time = time + 1
2. u.d = time
3. u.color = GRAY
4. **for each v ∈ G.Adj[u]**
5. **if v.color == WHITE**
6. v.π = u
7. DFS-VISIT(G, v)
8. u.color = BLACK
9. time = time + 1
10. u.f = time

New Version of DFS (co

time = 9

u.d: discovery time
u.f: finish time
u.d / u.f



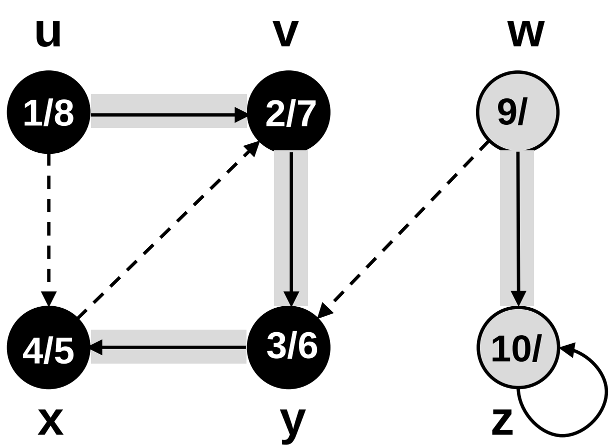

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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) // recursion
u.color = BLACK
time = time + 1
u.f = time
```

New Version of DFS (cont.)

time = 10

u.d: discovery time
u.f: finish time
u.d / u.f



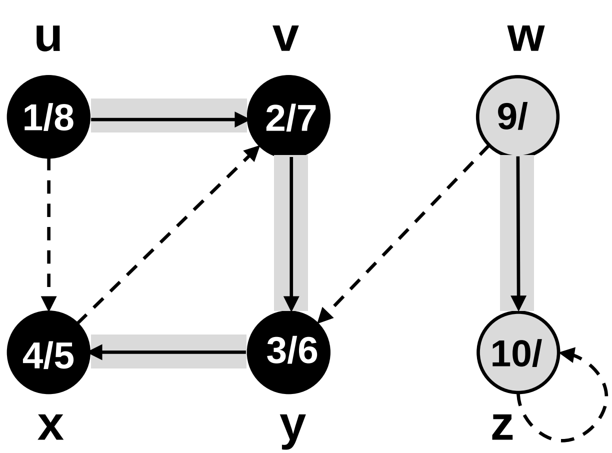

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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
```

New Version of DFS (co

time = 10

u.d: discovery time
u.f: finish time
u.d / u.f



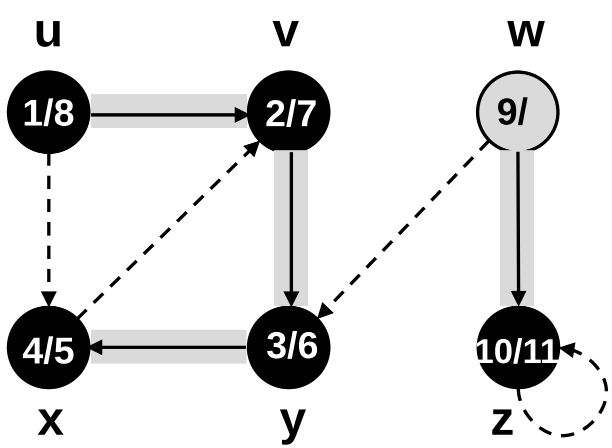

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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
```

New Version of DFS (cor

time = 11

u.d: discovery time
u.f: finish time
u.d / u.f



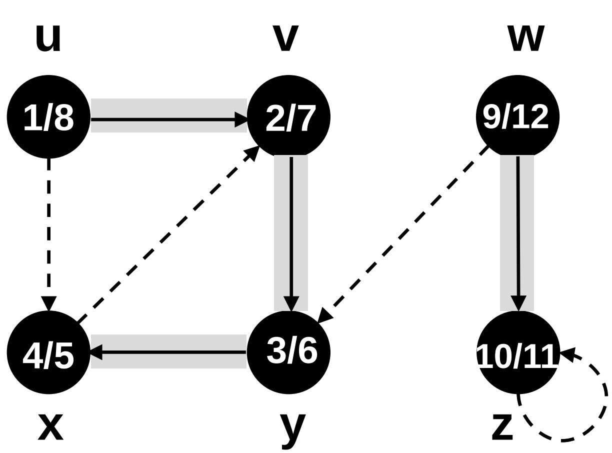

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
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
```

New Version of DFS (co

time = 12

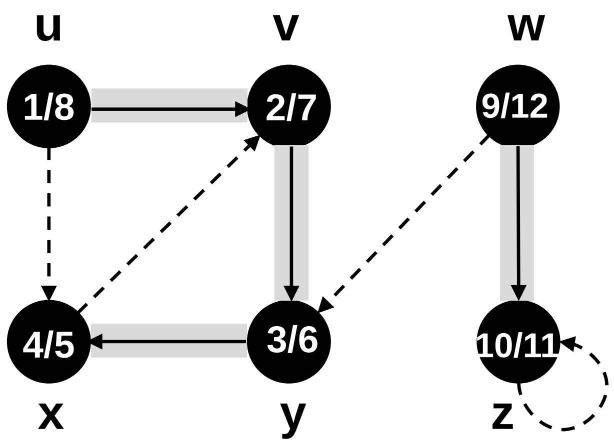
u.d: discovery time
u.f: finish time
u.d / u.f



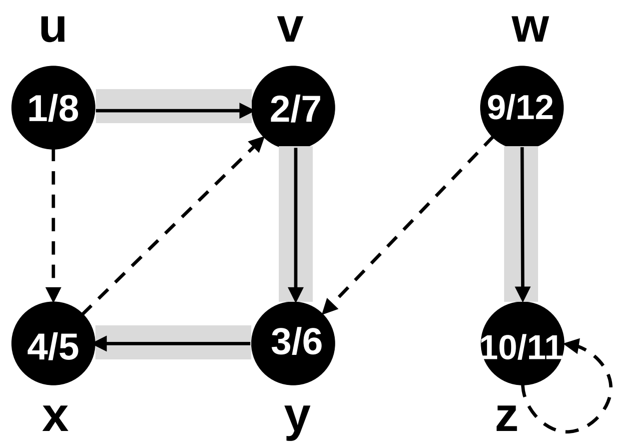

```
DFS(G)
1. for each vertex u ∈ G.V
2.     u.color = WHITE
3.     u.π = NIL
4. time = 0
5. for each vertex u ∈ G.V
6.     if u.color == WHITE
7.         DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
1. time = time + 1
2. u.d = time
3. u.color = GRAY
4. for each v ∈ G.Adj[u]
5.     if v.color == WHITE
6.         v.π = u
7.         DFS-VISIT(G, v)
8. u.color = BLACK
9. time = time + 1
10. u.f = time
```

New Version of DFS (cont.)



Topological Sort Algorithm



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