Graphs - DFS & Connected Components

Fundamental Algorithms

Rodica Potolea, Camelia Lemnaru and Ciprian Oprișa

Technical University of Cluj-Napoca
Computer Science Department

Agenda

1. Depth First Search (DFS)
2. Connected components
   - Connected components
   - Strongly connected components

Agenda

1. Depth First Search (DFS)

2. Connected components
   - Connected components
   - Strongly connected components

Depth First Search (1)

- similar to BFS, the queue is replaced by a stack
  - the stack is implicit with recursive calls
- instead of enqueuing the neighbors, we make a recursive call for each of them
- keep the color representation
  - WHITE - unvisited (all vertices are WHITE at first)
  - GRAY - under visitation (all GRAY vertices are on the stack)
  - BLACK - visited (in the end end, all nodes become BLACK)
- the color clusters define a vertical boundary between visited/unvisited vertices

Depth First Search (2)

- keep the parent attribute (π) - a reference to the vertex from where we reached the current vertex
- add the attributes
  - *d* - **d**iscovery time
  - *f* - **f**inish time
- π, *d* and *f* provide useful information for other algorithms
  - edge classification
  - topological sort

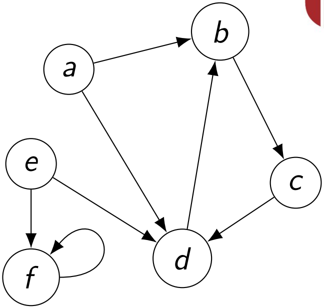
The DFS Algorithm

```
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
```

The DFS Algorithm

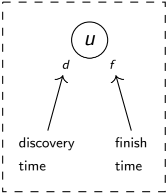
```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```




```
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

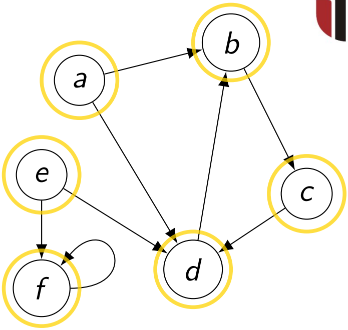
forest:



The DFS Algorithm

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)
```

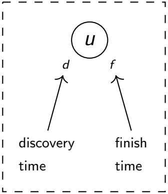


DFS-VISIT(G, u)

```
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

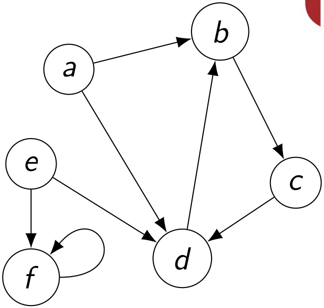
forest:



The DFS Algorithm

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)
```



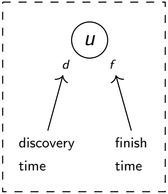
DFS-VISIT(G, u)

```
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
```

$time = 0$

DFS

forest:



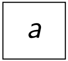
The DFS Algorithm

DFS(G)

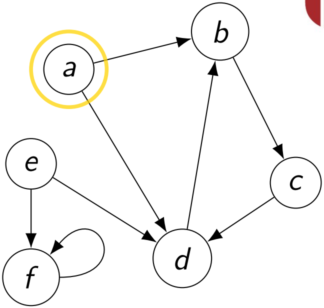
```
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

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
```



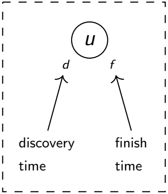
stack



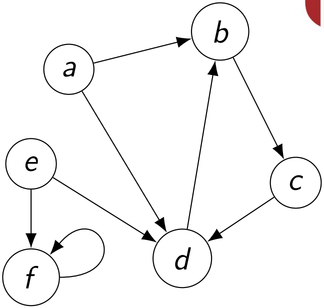
$time = 0$

DFS

forest:



The DFS Algorithm



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
```

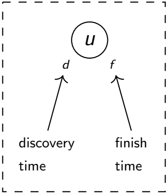
a

stack

$time = 1$

DFS

forest:



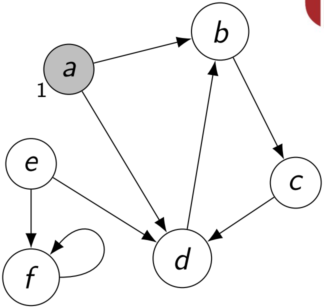
The DFS Algorithm

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
```



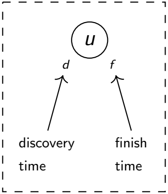
a

stack

$time = 1$

DFS forest:

a



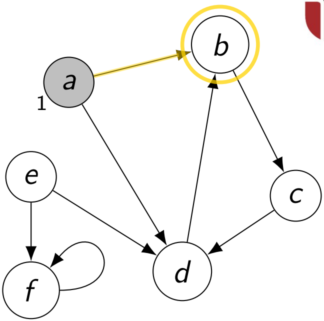
The DFS Algorithm

DFS(G)

for each vertex $u \in G.V$
  $u.color = \text{WHITE}$
  $u.\pi = \text{NIL}$
$time = 0$
for each vertex $u \in G.V$
  if $u.color == \text{WHITE}$
    DFS-VISIT(G, u)

DFS-VISIT(G, u)

$time = time + 1$
$u.d = time$
$u.color = \text{GRAY}$
for each $v \in G.Adj[u]$
  if $v.color == \text{WHITE}$
    $v.\pi = u$
    DFS-VISIT(G, v)
$u.color = \text{BLACK}$
$time = time + 1$
$u.f = time$

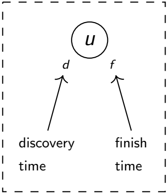


a

stack

$time = 1$

DFS forest:



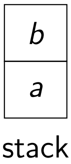
The DFS Algorithm

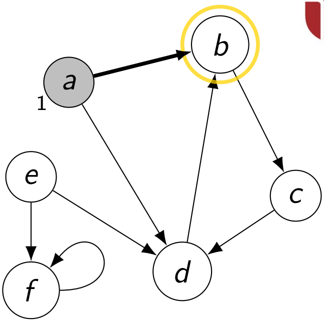
DFS(G)

for each vertex $u \in G.V$
    $u.color = \text{WHITE}$
    $u.\pi = \text{NIL}$
$time = 0$
for each vertex $u \in G.V$
    if $u.color == \text{WHITE}$
        DFS-VISIT(G, u)

DFS-VISIT(G, u)

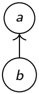
$time = time + 1$
$u.d = time$
$u.color = \text{GRAY}$
for each $v \in G.Adj[u]$
    if $v.color == \text{WHITE}$
        $v.\pi = u$
        DFS-VISIT(G, v)
$u.color = \text{BLACK}$
$time = time + 1$
$u.f = time$

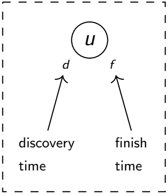




$time = 1$

DFS forest:





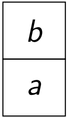
The DFS Algorithm

DFS(G)

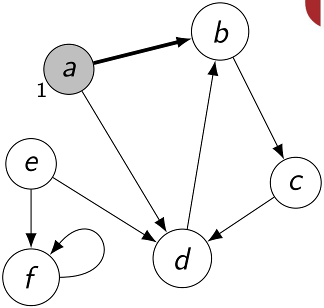
for each vertex $u \in G.V$
  $u.color = \text{WHITE}$
  $u.\pi = \text{NIL}$
$time = 0$
for each vertex $u \in G.V$
  if $u.color == \text{WHITE}$
    DFS-VISIT(G, u)

DFS-VISIT(G, u)

$time = time + 1$
$u.d = time$
$u.color = \text{GRAY}$
for each $v \in G.Adj[u]$
  if $v.color == \text{WHITE}$
    $v.\pi = u$
    DFS-VISIT(G, v)
$u.color = \text{BLACK}$
$time = time + 1$
$u.f = time$

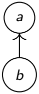


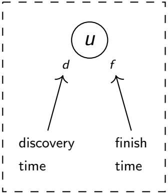
stack



$time = 2$

DFS forest:

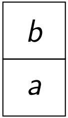




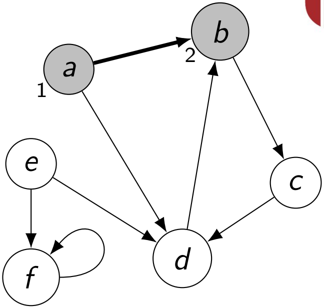
The DFS Algorithm

```
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
```

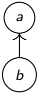


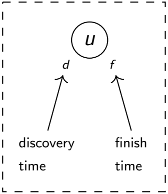
stack



$time = 2$

DFS forest:

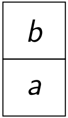




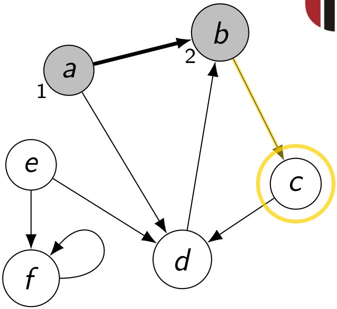
The DFS Algorithm

```
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
```

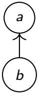


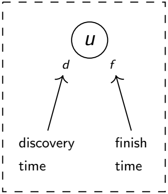
stack



$time = 2$

DFS forest:

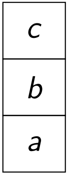




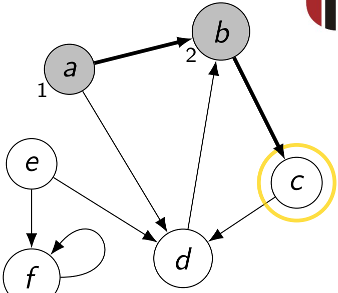
The DFS Algorithm

```
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
```

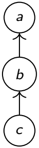


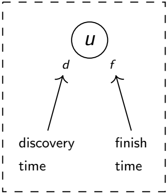
stack



$time = 2$

DFS forest:





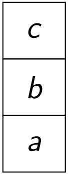
The DFS Algorithm

DFS(G)

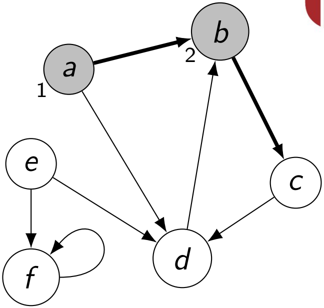
```
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

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
```

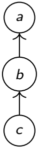


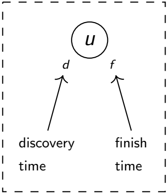
stack



$time = 3$

DFS forest:

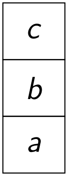




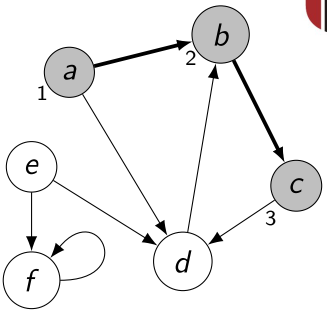
The DFS Algorithm

```
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
```

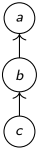


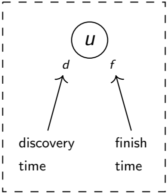
stack



$time = \ 3$

DFS forest:





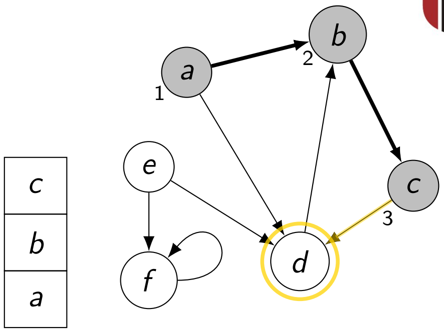
The DFS Algorithm

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)

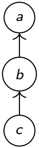
```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

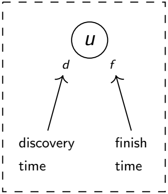


stack

$time = 3$

DFS forest:





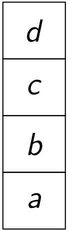
The DFS Algorithm

DFS(G)

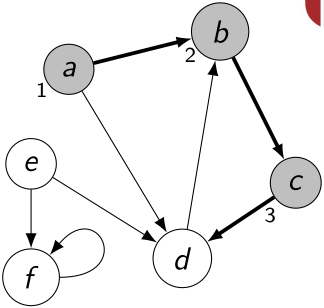
```
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

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
```

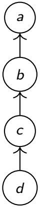


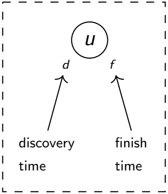
stack



$time = 3$

DFS forest:





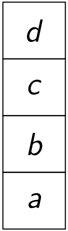
The DFS Algorithm

DFS(G)

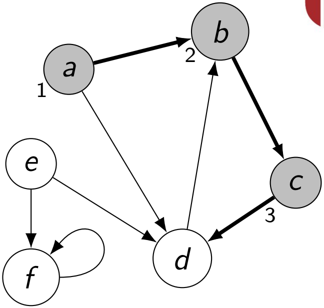
```
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

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
```

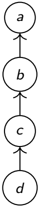


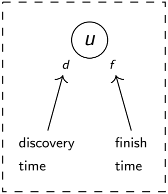
stack



$time = 4$

DFS forest:





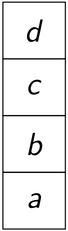
The DFS Algorithm

DFS(G)

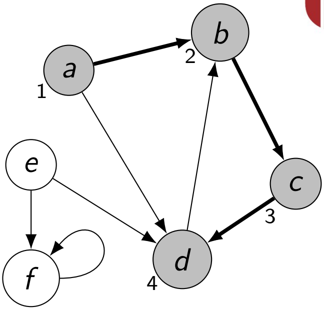
```
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

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
```

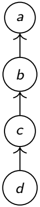


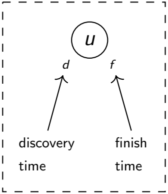
stack



$time = 4$

DFS forest:





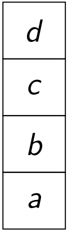
The DFS Algorithm

DFS(G)

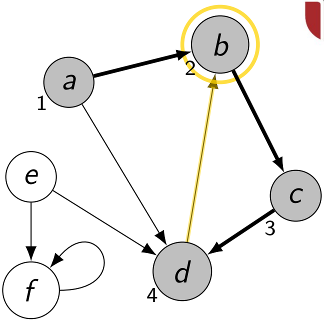
```
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

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
```

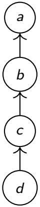


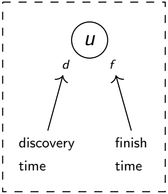
stack



$time = 4$

DFS forest:





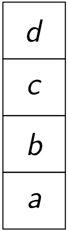
The DFS Algorithm

DFS(G)

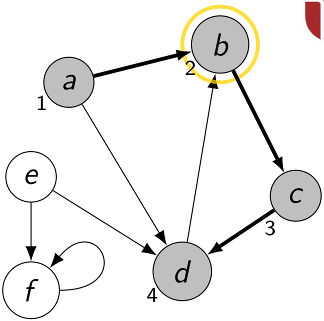
```
for each vertex u ∈ G.V
        u.color = WHITE
        u.π = NIL
time = 0
for each vertex u ∈ G.V
        if u.color == WHITE
                DFS-VISIT(G, u)
```

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
```

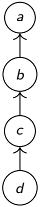


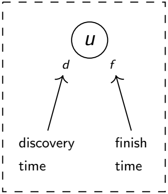
stack



$time = 4$

DFS forest:





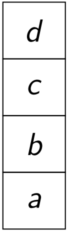
The DFS Algorithm

DFS(G)

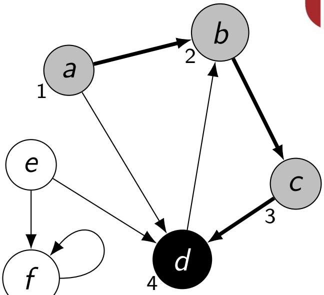
```
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

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
```

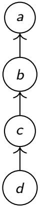


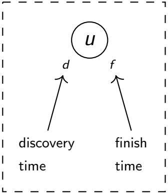
stack



$time = 4$

DFS forest:

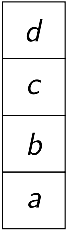




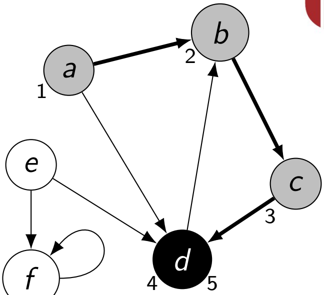
The DFS Algorithm

```
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
```

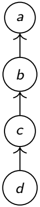


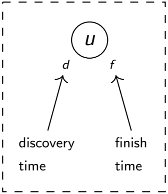
stack



$time = 5$

DFS forest:

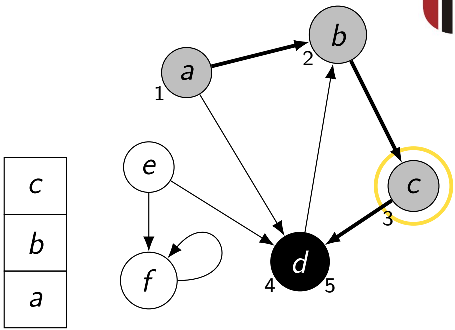




The DFS Algorithm

```
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)
```

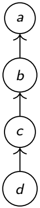
```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

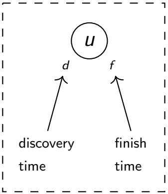


stack

$time = 5$

DFS forest:





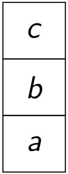
The DFS Algorithm

DFS(G)

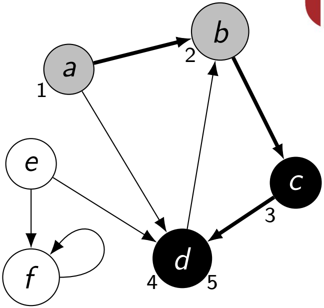
```
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

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
```

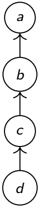


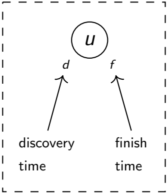
stack



$time = 5$

DFS forest:

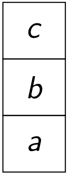




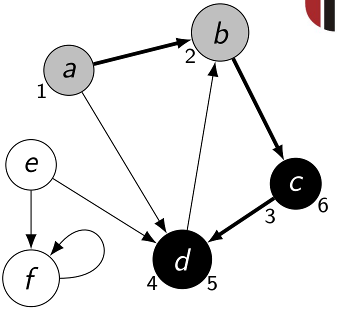
The DFS Algorithm

```
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
```

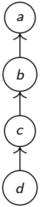


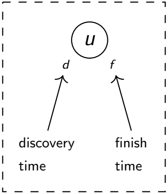
stack



$time = 6$

DFS forest:

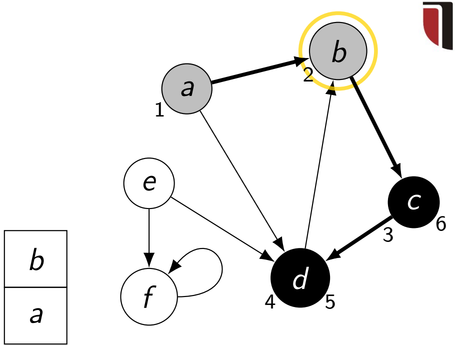




The DFS Algorithm

```
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)
```

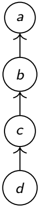
```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
        if v.color == WHITE
                v.π = u
                DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

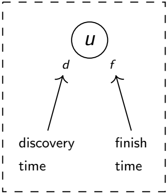


stack

$time = 6$

DFS forest:





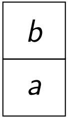
The DFS Algorithm

DFS(G)

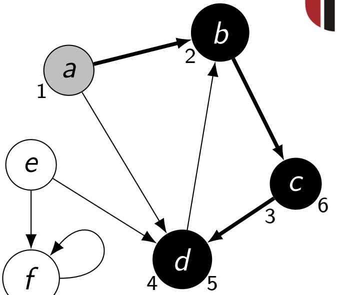
```
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

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
```

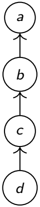


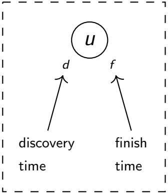
stack



$time = 6$

DFS forest:

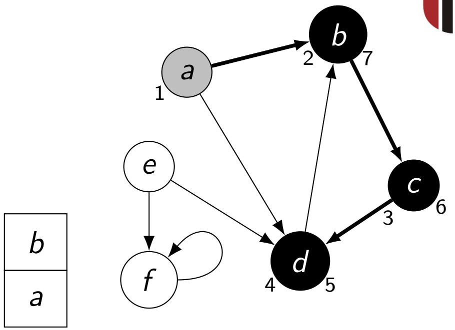




The DFS Algorithm

```
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)
```

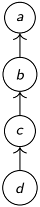
```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

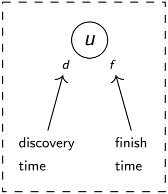


stack

$time = 7$

DFS forest:





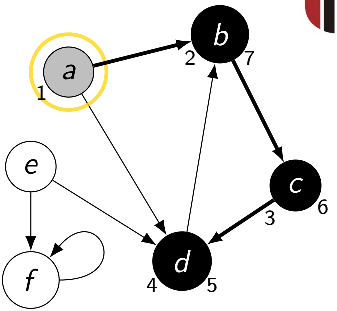
The DFS Algorithm

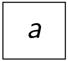
DFS(G)

```
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
```

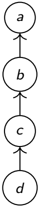


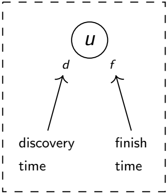


stack

$time = 7$

DFS forest:





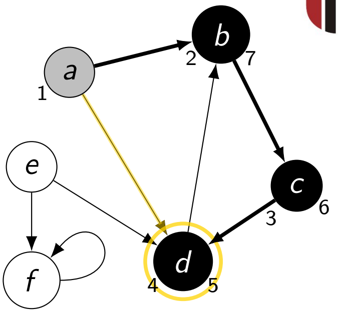
The DFS Algorithm

DFS(G)

for each vertex $u \in G.V$
    $u.color = \text{WHITE}$
    $u.\pi = \text{NIL}$
$time = 0$
for each vertex $u \in G.V$
    if $u.color == \text{WHITE}$
        DFS-VISIT(G, u)

DFS-VISIT(G, u)

$time = time + 1$
$u.d = time$
$u.color = \text{GRAY}$
for each $v \in G.Adj[u]$
    if $v.color == \text{WHITE}$
        $v.\pi = u$
        DFS-VISIT(G, v)
$u.color = \text{BLACK}$
$time = time + 1$
$u.f = time$

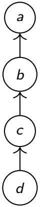


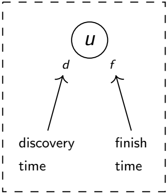
a

stack

$time = 7$

DFS forest:





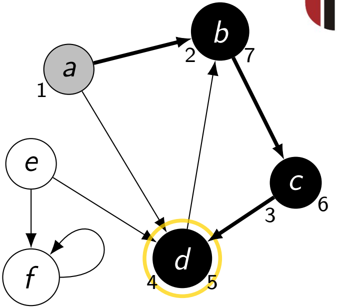
The DFS Algorithm

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
```

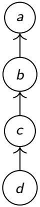


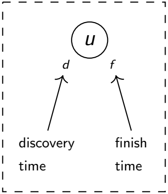
a

stack

$time = 7$

DFS forest:





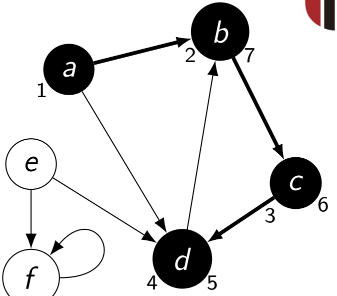
The DFS Algorithm

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
```

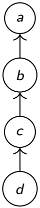


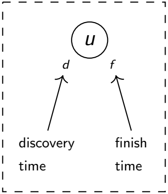
$\boxed{a}$

stack

$time = 7$

DFS forest:





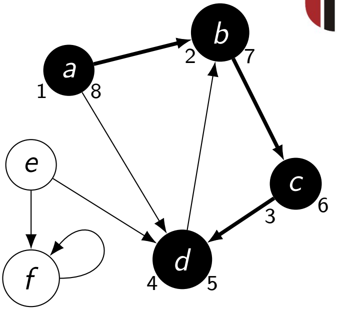
The DFS Algorithm

DFS(G)

for each vertex $u \in G.V$
    $u.color = \text{WHITE}$
    $u.\pi = \text{NIL}$
$time = 0$
for each vertex $u \in G.V$
    if $u.color == \text{WHITE}$
        DFS-VISIT(G, u)

DFS-VISIT(G, u)

$time = time + 1$
$u.d = time$
$u.color = \text{GRAY}$
for each $v \in G.Adj[u]$
    if $v.color == \text{WHITE}$
        $v.\pi = u$
        DFS-VISIT(G, v)
$u.color = \text{BLACK}$
$time = time + 1$
$u.f = time$

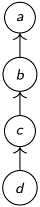


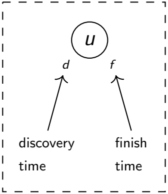
a

stack

$time = 8$

DFS forest:

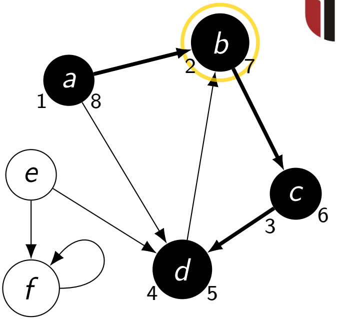




The DFS Algorithm

DFS(G)

for each vertex $u \in G.V$
    $u.color = \text{WHITE}$
    $u.\pi = \text{NIL}$
$time = 0$
for each vertex $u \in G.V$
    if $u.color == \text{WHITE}$
        DFS-VISIT(G, u)



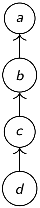
DFS-VISIT(G, u)

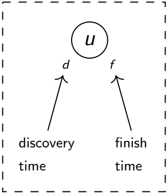
$time = time + 1$
$u.d = time$
$u.color = \text{GRAY}$
for each $v \in G.Adj[u]$
    if $v.color == \text{WHITE}$
        $v.\pi = u$
        DFS-VISIT(G, v)
$u.color = \text{BLACK}$
$time = time + 1$
$u.f = time$

stack

$time = 8$

DFS forest:

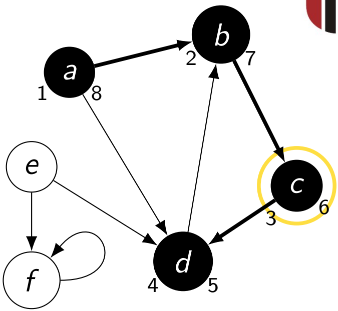




The DFS Algorithm

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)
```



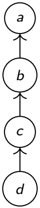
DFS-VISIT(G, u)

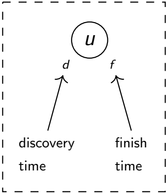
```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

stack

$time = 8$

DFS forest:

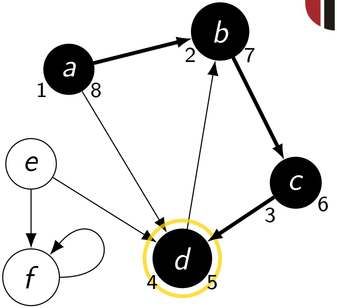




The DFS Algorithm

DFS(G)

for each vertex $u \in G.V$
    $u.color = \text{WHITE}$
    $u.\pi = \text{NIL}$
$time = 0$
for each vertex $u \in G.V$
    if $u.color == \text{WHITE}$
        DFS-VISIT(G, u)



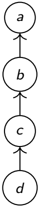
DFS-VISIT(G, u)

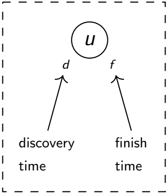
$time = time + 1$
$u.d = time$
$u.color = \text{GRAY}$
for each $v \in G.Adj[u]$
    if $v.color == \text{WHITE}$
        $v.\pi = u$
        DFS-VISIT(G, v)
$u.color = \text{BLACK}$
$time = time + 1$
$u.f = time$

stack

$time = 8$

DFS forest:

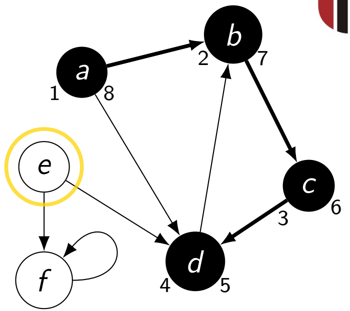




The DFS Algorithm

```
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
```

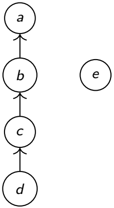


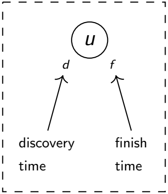
e

stack

$time = 8$

DFS forest:

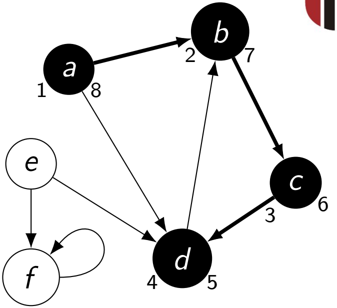




The DFS Algorithm

```
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
```

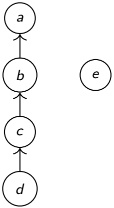


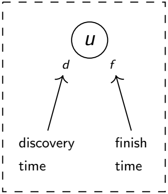
e

stack

$time = 9$

DFS forest:





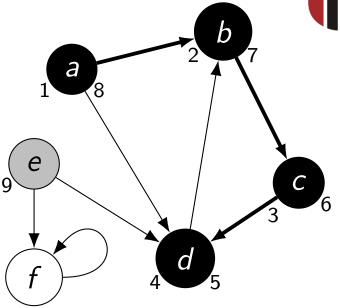
The DFS Algorithm

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
```

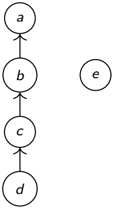


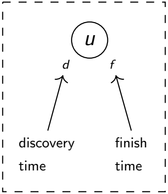
e

stack

$time = 9$

DFS forest:





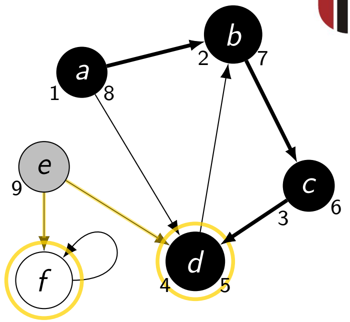
The DFS Algorithm

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
```

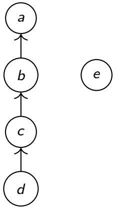


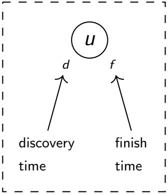
e

stack

$time = 9$

DFS forest:





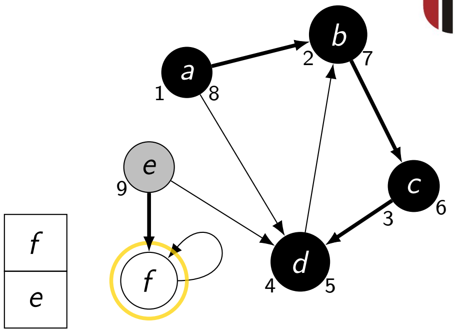
The DFS Algorithm

$\text{DFS}(G)$

```
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)
```

$\text{DFS-VISIT}(G, u)$

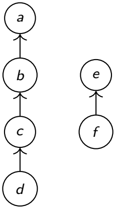
```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

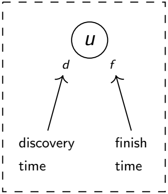


stack

$time = 9$

DFS forest:





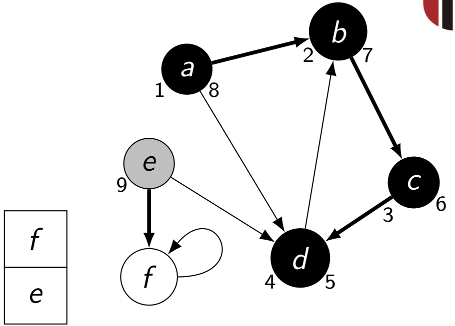
The DFS Algorithm

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)

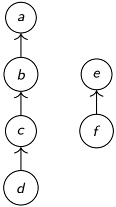
```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

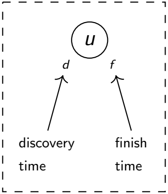


stack

$time = 10$

DFS forest:





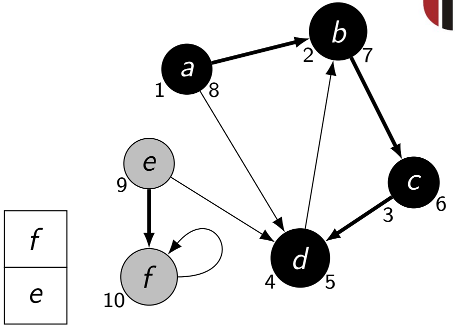
The DFS Algorithm

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)

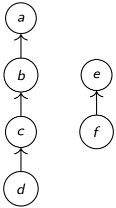
```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

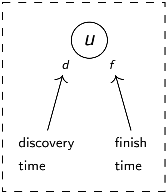


stack

$time = 10$

DFS forest:





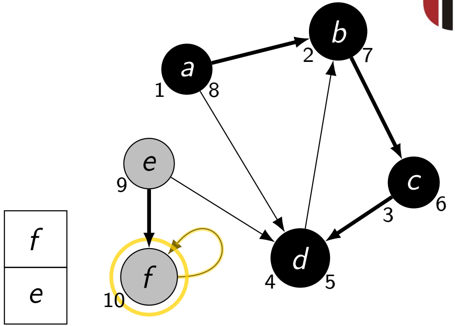
The DFS Algorithm

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)

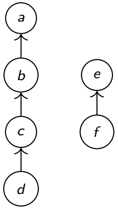
```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

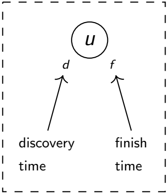


stack

$time = 10$

DFS forest:





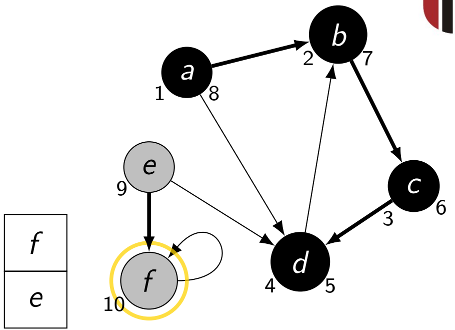
The DFS Algorithm

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)

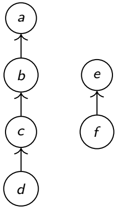
```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

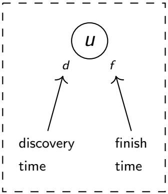


stack

$time = 10$

DFS forest:





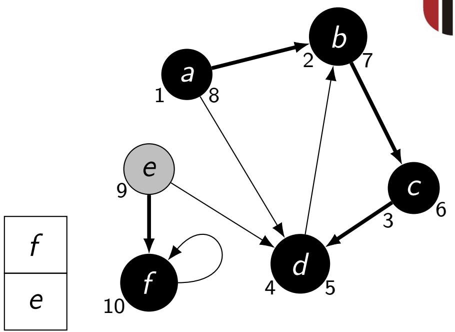
The DFS Algorithm

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)

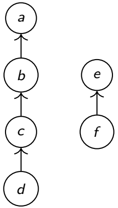
```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

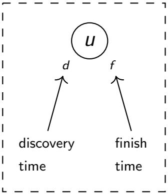


stack

$time = 10$

DFS forest:





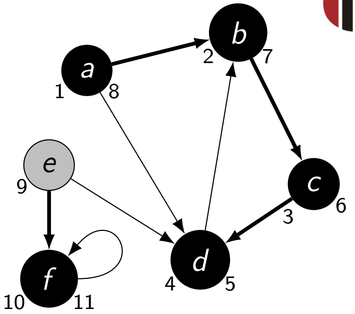
The DFS Algorithm

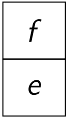
DFS(G)

```
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
```

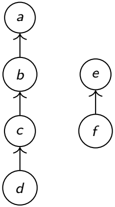


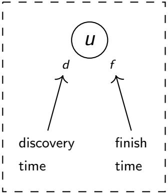


stack

$time = 11$

DFS forest:





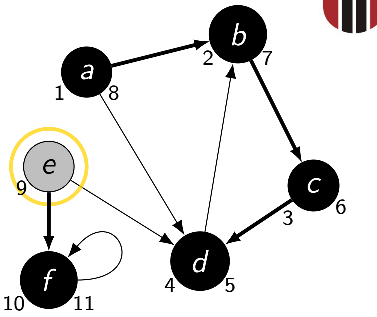
The DFS Algorithm

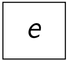
DFS(G)

```
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
```

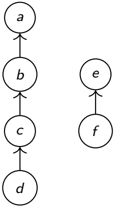


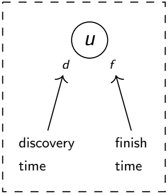


stack

$time = 11$

DFS forest:





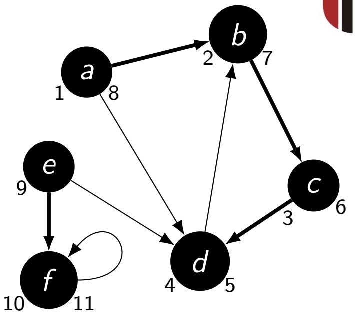
The DFS Algorithm

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
```

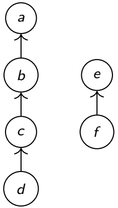


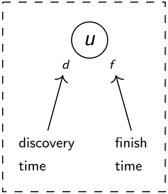
e

stack

$time = 11$

DFS forest:





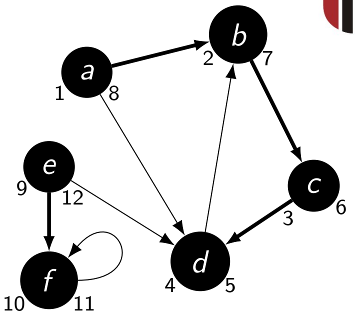
The DFS Algorithm

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
```

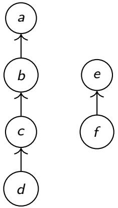


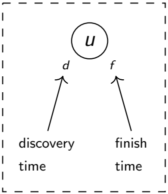
e

stack

$time = 12$

DFS forest:

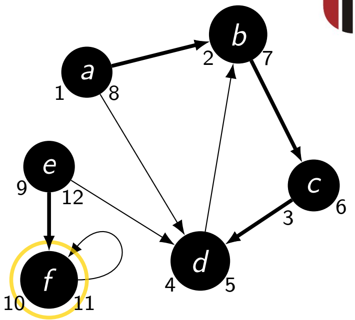




The DFS Algorithm

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)
```



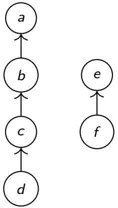
DFS-VISIT(G, u)

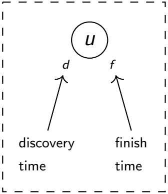
```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

stack

$time = 12$

DFS forest:

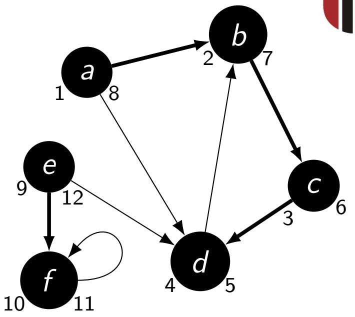




The DFS Algorithm

DFS(G)

for each vertex $u \in G.V$
  $u.color = \text{WHITE}$
  $u.\pi = \text{NIL}$
$time = 0$
for each vertex $u \in G.V$
  if $u.color == \text{WHITE}$
    DFS-VISIT(G, u)



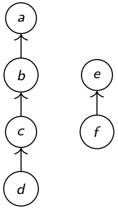
DFS-VISIT(G, u)

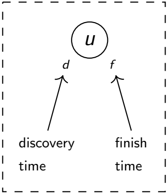
$time = time + 1$
$u.d = time$
$u.color = \text{GRAY}$
for each $v \in G.Adj[u]$
  if $v.color == \text{WHITE}$
    $v.\pi = u$
    DFS-VISIT(G, v)
$u.color = \text{BLACK}$
$time = time + 1$
$u.f = time$

stack

$time = 12$

DFS forest:





DFS Algorithm - Complexity

```
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 Algorithm - Complexity

```
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)
```

$O(V)$

```
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 Algorithm - Complexity

```
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)
```

$O(V)$

$O(V)$

```
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 Algorithm - Complexity

```
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)
```

$O(V)$

$O(V)$

called $|V|$ times

```
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 Algorithm - Complexity

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

```
DFS-VISIT(G, u)                      called |V| times
time = time + 1                      } O(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                      } O(1)
time = time + 1
u.f = time
```

DFS Algorithm - Complexity

DFS(G)

for each vertex $u \in G.V$
  $u.color = \text{WHITE}$
  $u.\pi = \text{NIL}$

$O(V)$

$time = 0$
for each vertex $u \in G.V$
  if $u.color == \text{WHITE}$
    DFS-VISIT(G, u)

$O(V)$

called $|V|$ times

DFS-VISIT(G, u)

$time = time + 1$
$u.d = time$
$u.color = \text{GRAY}$

$O(1)$

for each $v \in G.Adj[u]$
  if $v.color == \text{WHITE}$
    $v.\pi = u$
    DFS-VISIT(G, v)

each neighbor
for every vertex,
so $|E|$ steps in total

$u.color = \text{BLACK}$
$time = time + 1$
$u.f = time$

$O(1)$

DFS Algorithm - Complexity

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)
```

$O(V)$

$O(V)$

called $|V|$ times

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
```

$O(1)$

each neighbor
for every vertex,
so $|E|$ steps in total

$O(1)$

$O(1)$

DFS Algorithm - Complexity

Complexity: $O(V+E)$

```
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)
```

$O(V)$

$O(V)$

```
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
```

called $|V|$ times

$O(1)$

each neighbor for every vertex, so $|E|$ steps in total

$O(1)$

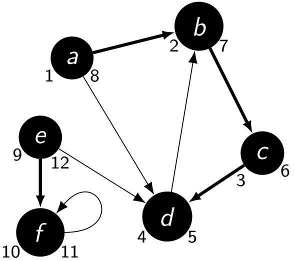
$O(1)$

Parenthesis theorem

For any graph $G = (V, E)$ and for any $u, v \in G.V$, exactly one of the three conditions hold:

- $[u.d, u.f] \cap [v.d, v.f] = \emptyset$
- $[u.d, u.f] \subset [v.d, v.f]$
- $[v.d, v.f] \subset [u.d, u.f]$

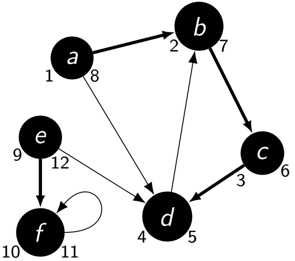
Parenthesis theorem



For any graph $G = (V, E)$ and for any $u, v \in G.V$, exactly one of the three conditions hold:

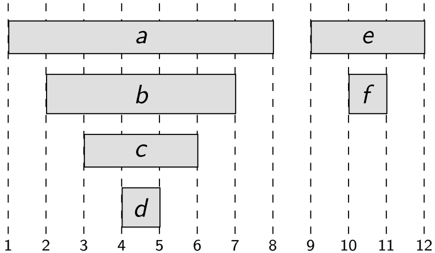
- $[u.d, u.f] \cap [v.d, v.f] = \emptyset$
- $[u.d, u.f] \subset [v.d, v.f]$
- $[v.d, v.f] \subset [u.d, u.f]$

Parenthesis theorem

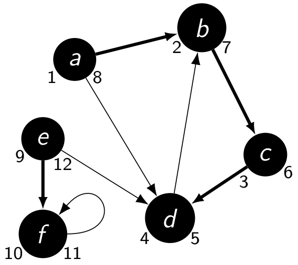


For any graph $G = (V, E)$ and for any $u, v \in G.V$, exactly one of the three conditions hold:

- $[u.d, u.f] \cap [v.d, v.f] = \emptyset$
- $[u.d, u.f] \subset [v.d, v.f]$
- $[v.d, v.f] \subset [u.d, u.f]$

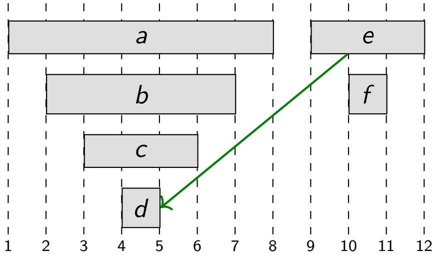


Parenthesis theorem

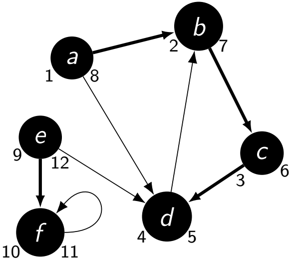


For any graph $G = (V, E)$ and for any $u, v \in G.V$, exactly one of the three conditions hold:

- $[u.d, u.f] \cap [v.d, v.f] = \emptyset$
  neither u nor v is a descendant of the other
- $[u.d, u.f] \subset [v.d, v.f]$
- $[v.d, v.f] \subset [u.d, u.f]$

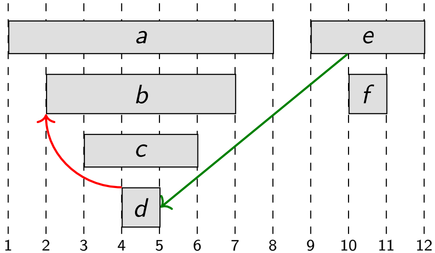


Parenthesis theorem

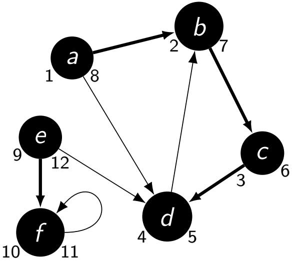


For any graph $G = (V, E)$ and for any $u, v \in G.V$, exactly one of the three conditions hold:

- $[u.d, u.f] \cap [v.d, v.f] = \emptyset$
  neither u nor v is a descendant of the other
- $[u.d, u.f] \subset [v.d, v.f]$
  u is a descendant of v
- $[v.d, v.f] \subset [u.d, u.f]$

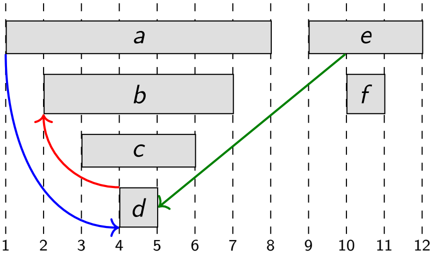


Parenthesis theorem



For any graph $G = (V, E)$ and for any $u, v \in G.V$, exactly one of the three conditions hold:

- $[u.d, u.f] \cap [v.d, v.f] = \emptyset$
  neither u nor v is a descendant of the other
- $[u.d, u.f] \subset [v.d, v.f]$
  u is a descendant of v
- $[v.d, v.f] \subset [u.d, u.f]$
  v is a descendant of u

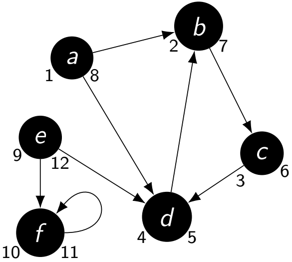


Edges characterization

- **Tree edges**
- Back edges
- Forward edges
- Cross edges

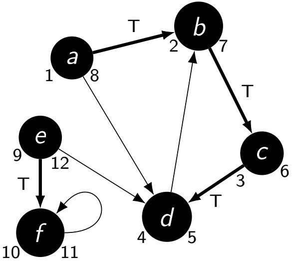
Edges characterization

- **Tree edges**
- Back edges
- Forward edges
- Cross edges

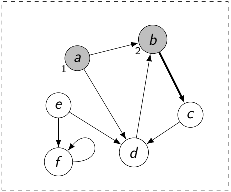


Edges characterization

- **Tree edges**
  - $u = \text{GRAY};\ v = \text{WHITE}$
  - $u.d < v.d < v.f < u.f$
- Back edges



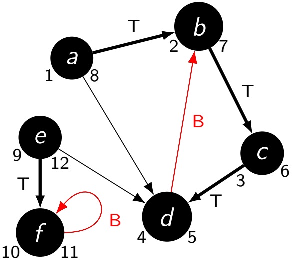
- Forward edges

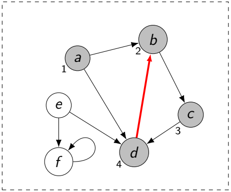


- Cross edges

Edges characterization

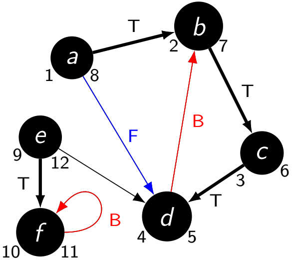
- **Tree edges**
  - $u = \text{GRAY};\ v = \text{WHITE}$
  - $u.d < v.d < v.f < u.f$
- Back edges
  - $u = \text{GRAY};\ v = \text{GRAY}$
  - $v.d < u.d < u.f < v.f$
  - also self-loops
- Forward edges
- Cross edges

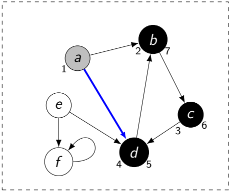




Edges characterization

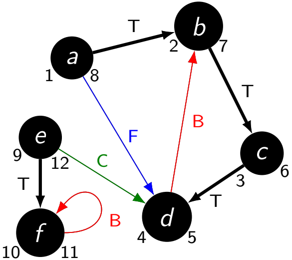
- **Tree edges**
  - $u = \text{GRAY};\ v = \text{WHITE}$
  - $u.d < v.d < v.f < u.f$
- Back edges
  - $u = \text{GRAY};\ v = \text{GRAY}$
  - $v.d < u.d < u.f < v.f$
  - also self-loops
- Forward edges
  - $u = \text{GRAY};\ v = \text{BLACK}$
  - $u.d < v.d < v.f < u.f$
- Cross edges

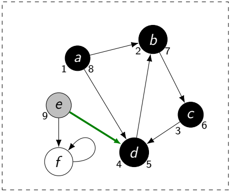




Edges characterization

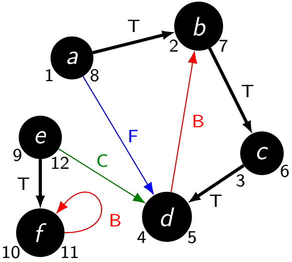
- **Tree edges**
  - $u = \text{GRAY}; \ v = \text{WHITE}$
  - $u.d < v.d < v.f < u.f$
- Back edges
  - $u = \text{GRAY}; \ v = \text{GRAY}$
  - $v.d < u.d < u.f < v.f$
  - also self-loops
- Forward edges
  - $u = \text{GRAY}; \ v = \text{BLACK}$
  - $u.d < v.d < v.f < u.f$
- Cross edges
  - $u = \text{GRAY}; \ v = \text{BLACK}$
  - $v.d < v.f < u.d < u.f$





Edges characterization

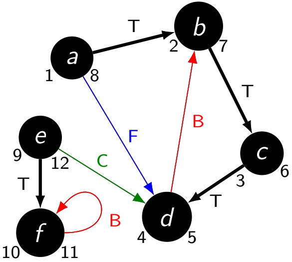
- **Tree edges**
  - $u = \text{GRAY}; \; v = \text{WHITE}$
  - $u.d < v.d < v.f < u.f$
- Back edges
  - $u = \text{GRAY}; \; v = \text{GRAY}$
  - $v.d < u.d < u.f < v.f$
  - also self-loops
- Forward edges
  - $u = \text{GRAY}; \; v = \text{BLACK}$
  - $u.d < v.d < v.f < u.f$
- Cross edges
  - $u = \text{GRAY}; \; v = \text{BLACK}$
  - $v.d < v.f < u.d < u.f$



Q: which edges produce cycles?

Edges characterization

- **Tree edges**
  - $u = \text{GRAY};\ v = \text{WHITE}$
  - $u.d < v.d < v.f < u.f$
- Back edges [they produce cycles]
  - $u = \text{GRAY};\ v = \text{GRAY}$
  - $v.d < u.d < u.f < v.f$
  - also self-loops
- Forward edges
  - $u = \text{GRAY};\ v = \text{BLACK}$
  - $u.d < v.d < v.f < u.f$
- Cross edges
  - $u = \text{GRAY};\ v = \text{BLACK}$
  - $v.d < v.f < u.d < u.f$



Edges characterization - undirected graphs

- which types of edges can we encounter in undirected graphs?

Edges characterization - undirected graphs

- which types of edges can we encounter in undirected graphs?

Theorem

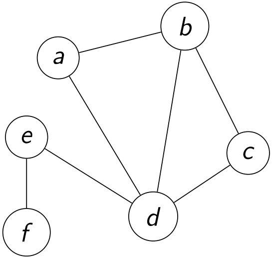
In a depth-first search of an undirected graph G, every edge of G is either a tree edge or a back edge.

Edges characterization - undirected graphs

- which types of edges can we encounter in undirected graphs?

Theorem

In a depth-first search of an undirected graph G, every edge of G is either a tree edge or a back edge.

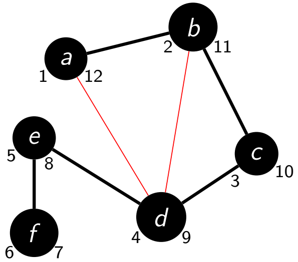


Edges characterization - undirected graphs

- which types of edges can we encounter in undirected graphs?

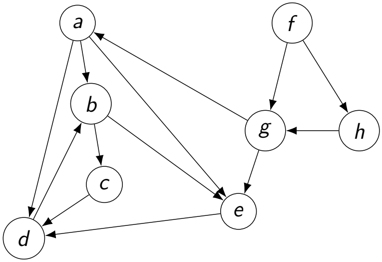
Theorem

In a depth-first search of an undirected graph G, every edge of G is either a tree edge or a back edge.



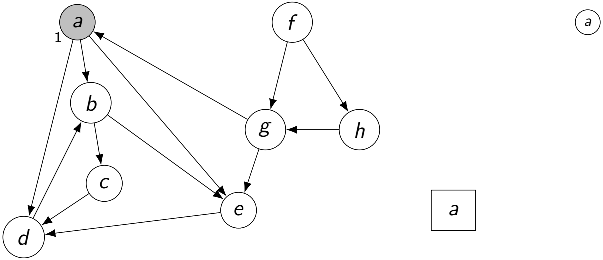
DFS vs. BFS example

DFS:



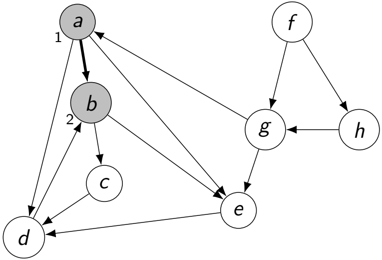
DFS vs. BFS example

DFS:

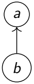


DFS vs. BFS example

DFS:

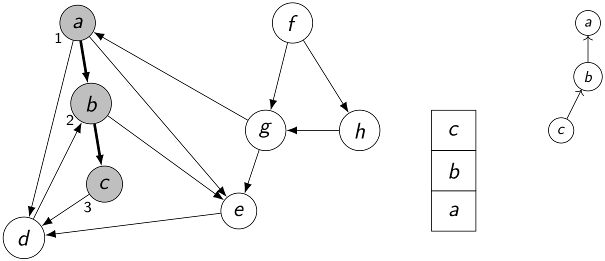


b
a



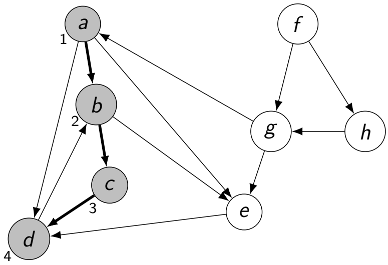
DFS vs. BFS example

DFS:

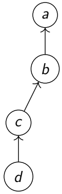


DFS vs. BFS example

DFS:

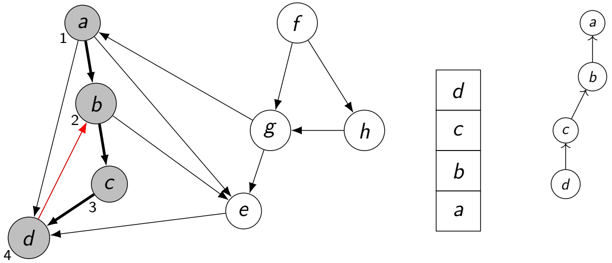


d
c
b
a



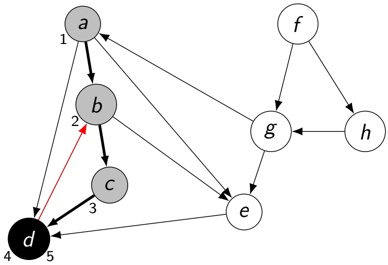
DFS vs. BFS example

DFS:

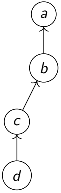


DFS vs. BFS example

DFS:

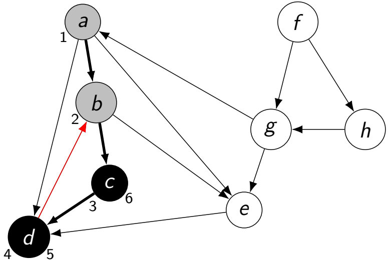


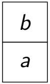
c
b
a

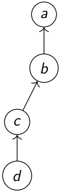


DFS vs. BFS example

DFS:

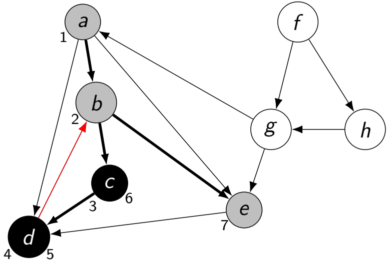


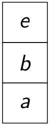


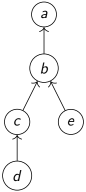


DFS vs. BFS example

DFS:

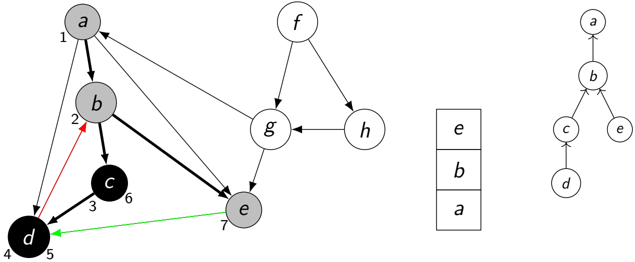






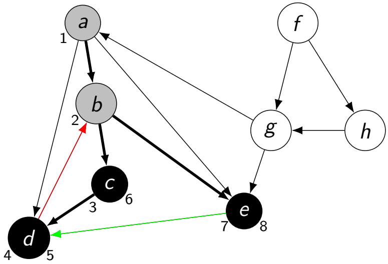
DFS vs. BFS example

DFS:

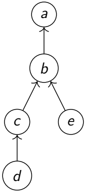


DFS vs. BFS example

DFS:

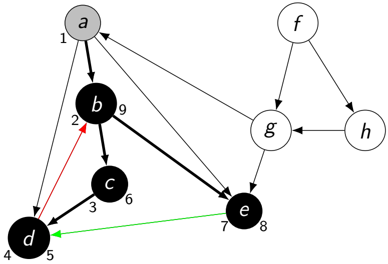


b
a

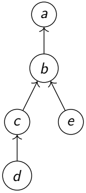


DFS vs. BFS example

DFS:

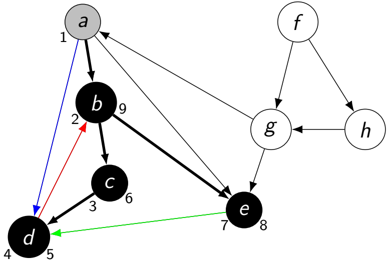


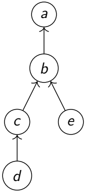
a



DFS vs. BFS example

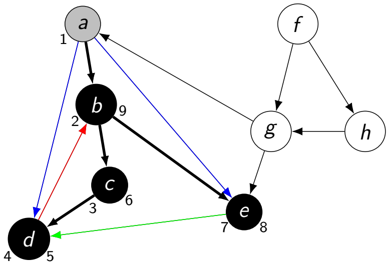
DFS:



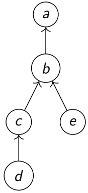


DFS vs. BFS example

DFS:

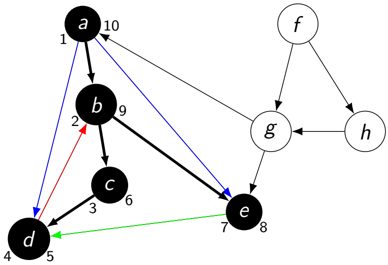


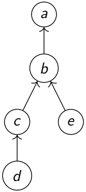
a



DFS vs. BFS example

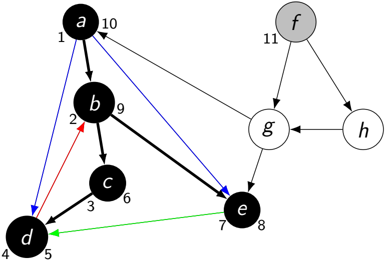
DFS:



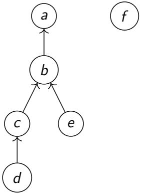


DFS vs. BFS example

DFS:

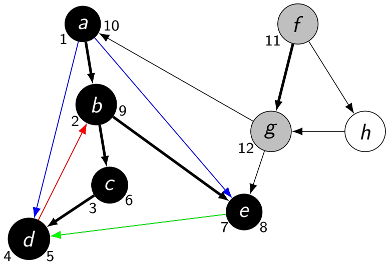


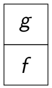
f

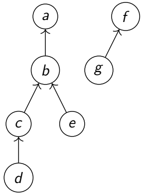


DFS vs. BFS example

DFS:

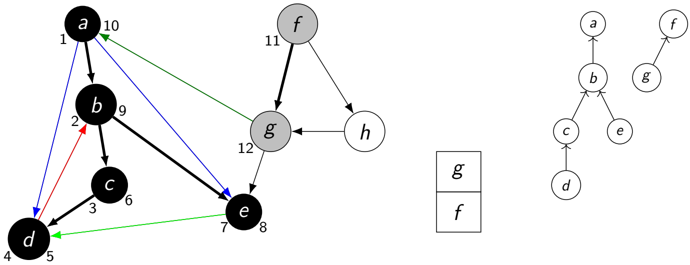






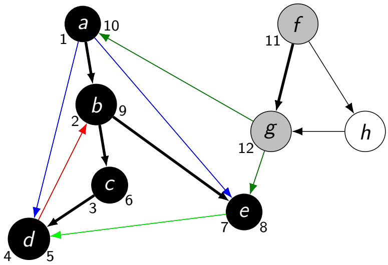
DFS vs. BFS example

DFS:

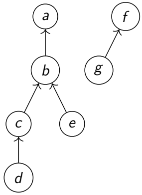


DFS vs. BFS example

DFS:

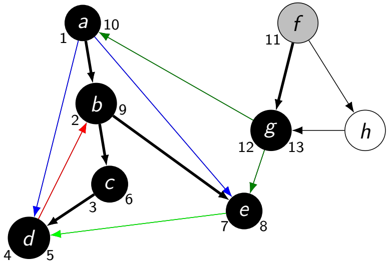


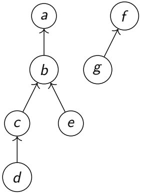
g
f



DFS vs. BFS example

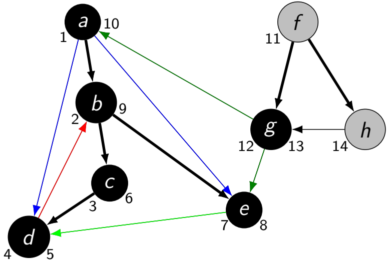
DFS:



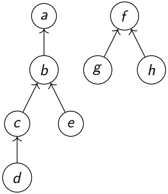


DFS vs. BFS example

DFS:

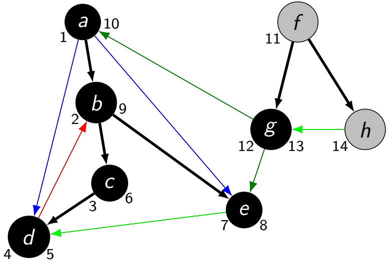


h
f

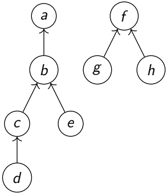


DFS vs. BFS example

DFS:

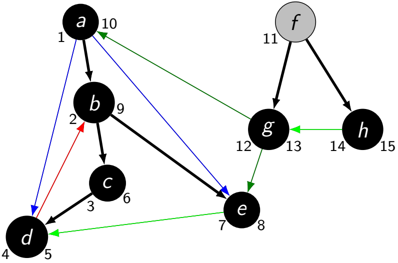


h
f

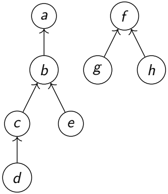


DFS vs. BFS example

DFS:

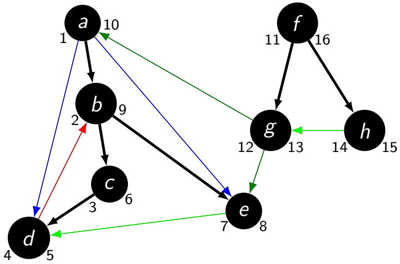


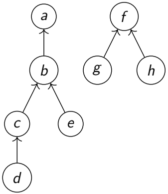
f



DFS vs. BFS example

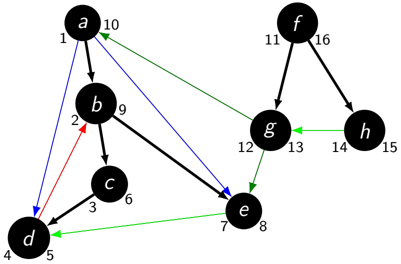
DFS:

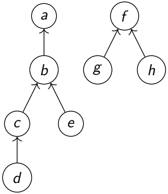




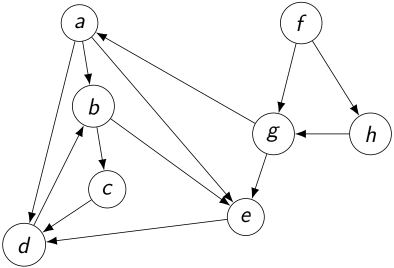
DFS vs. BFS example

DFS:



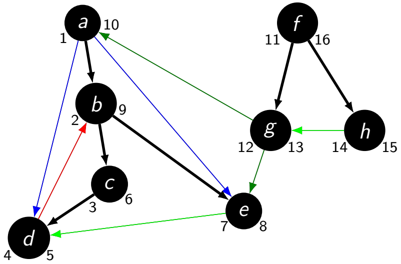


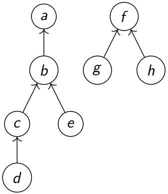
BFS:



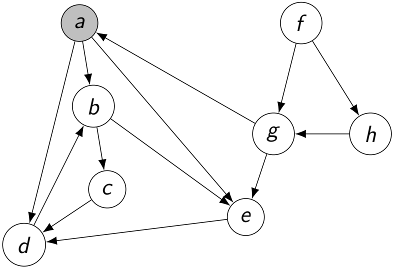
DFS vs. BFS example

DFS:



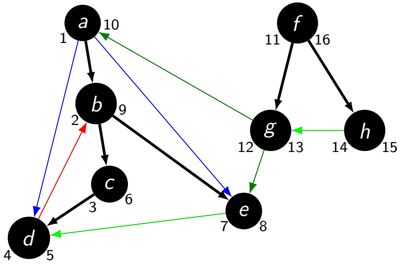


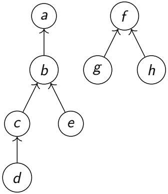
BFS:



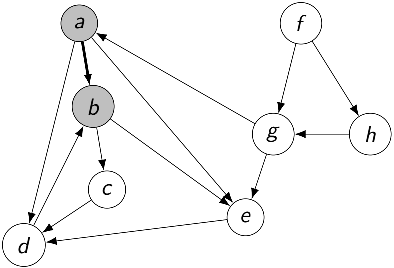
DFS vs. BFS example

DFS:

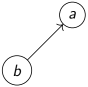




BFS:

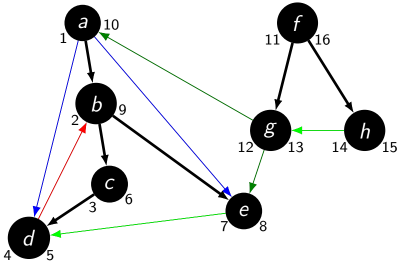


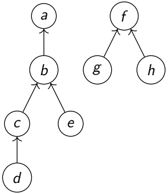
a	b



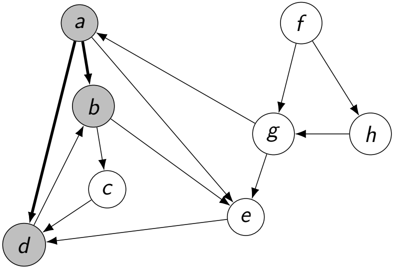
DFS vs. BFS example

DFS:

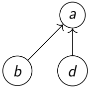




BFS:

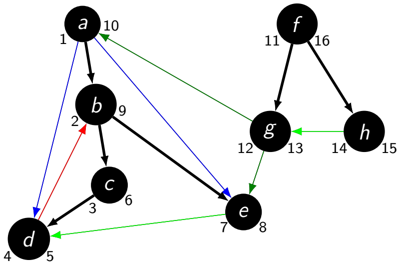


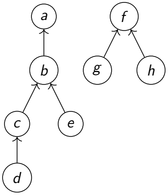
a	b	d



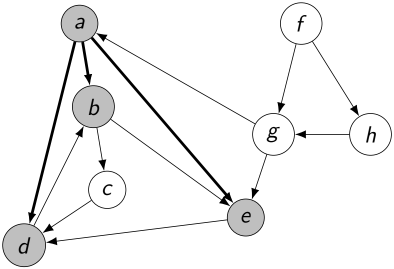
DFS vs. BFS example

DFS:

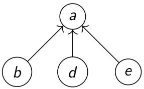




BFS:

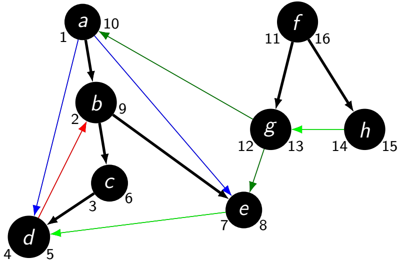


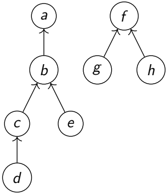
a	b	d	e



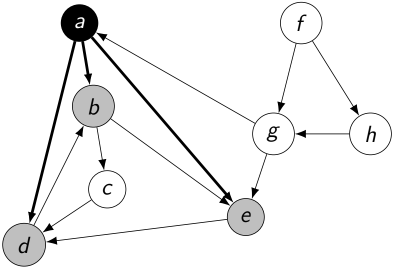
DFS vs. BFS example

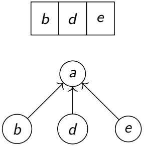
DFS:





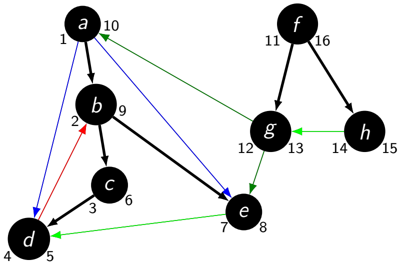
BFS:

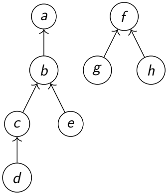




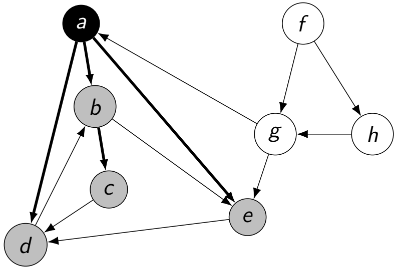
DFS vs. BFS example

DFS:

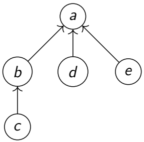




BFS:

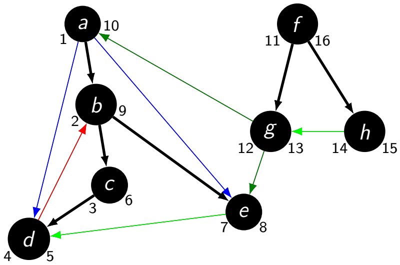


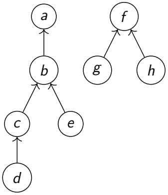
b	d	e	c



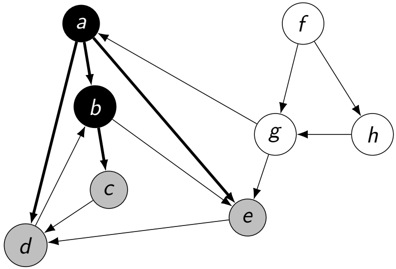
DFS vs. BFS example

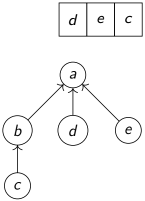
DFS:





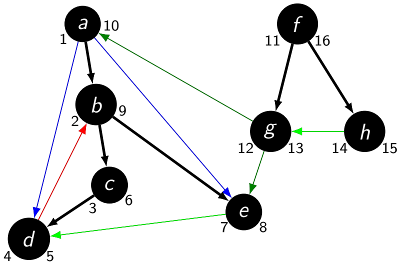
BFS:

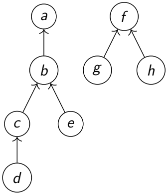




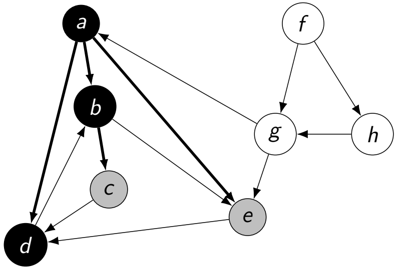
DFS vs. BFS example

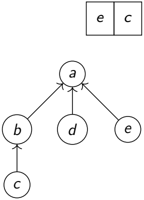
DFS:





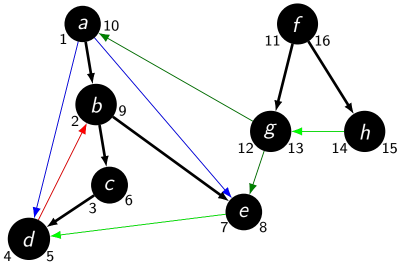
BFS:

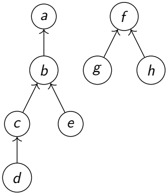




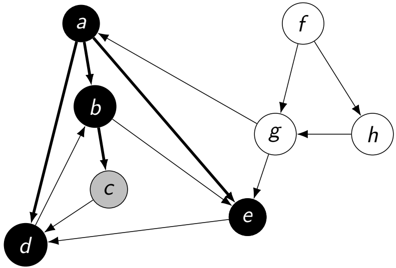
DFS vs. BFS example

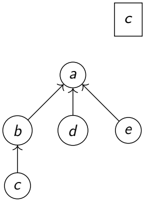
DFS:





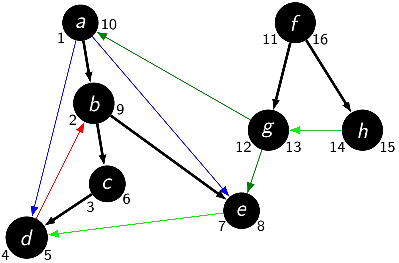
BFS:

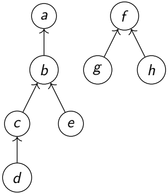




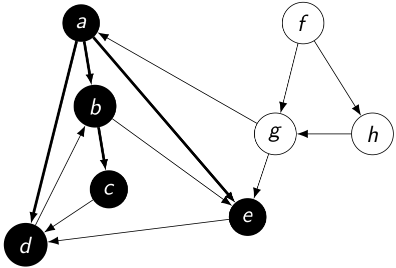
DFS vs. BFS example

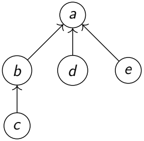
DFS:





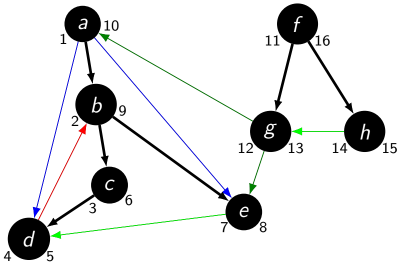
BFS:

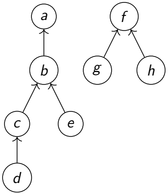




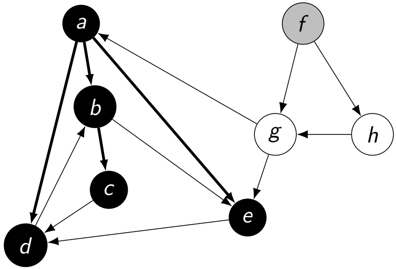
DFS vs. BFS example

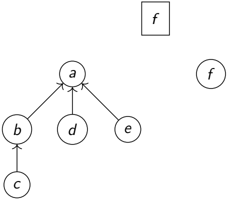
DFS:





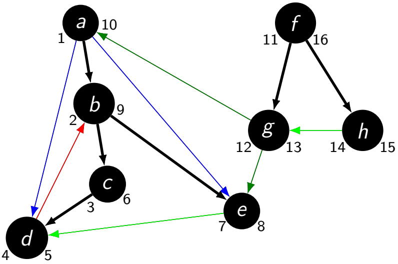
BFS:

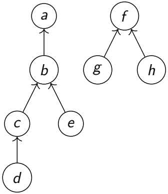




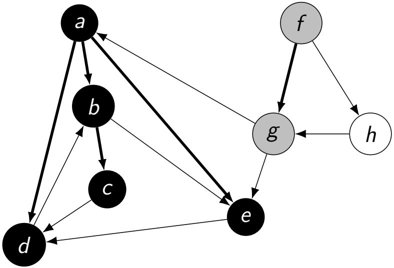
DFS vs. BFS example

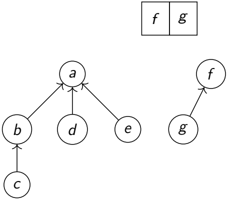
DFS:





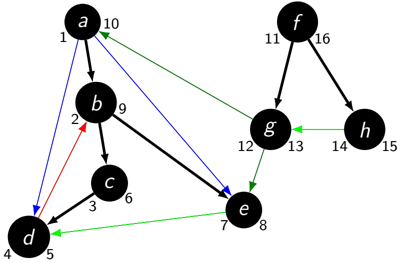
BFS:

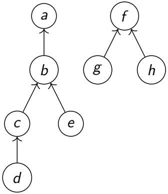




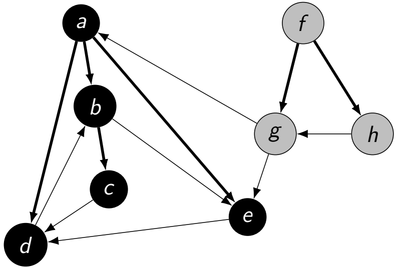
DFS vs. BFS example

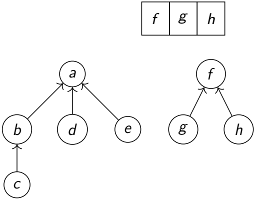
DFS:





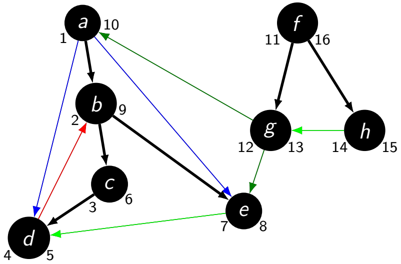
BFS:

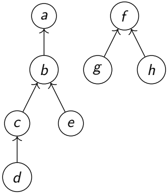




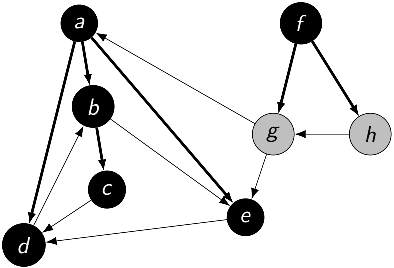
DFS vs. BFS example

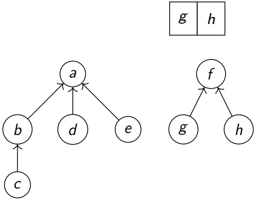
DFS:





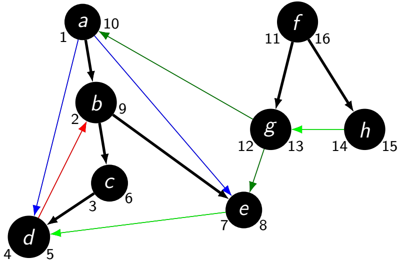
BFS:

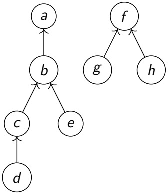




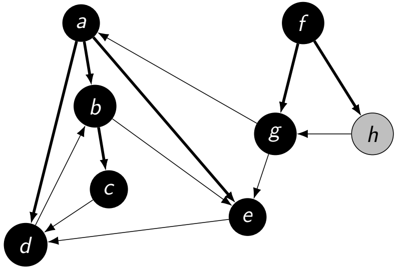
DFS vs. BFS example

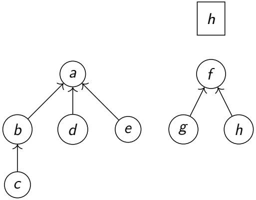
DFS:





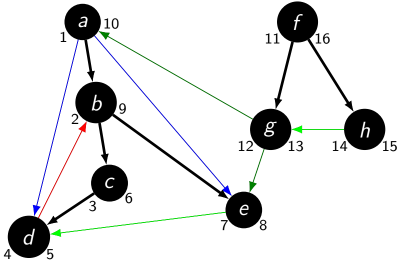
BFS:

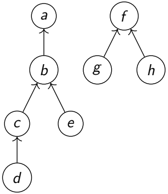




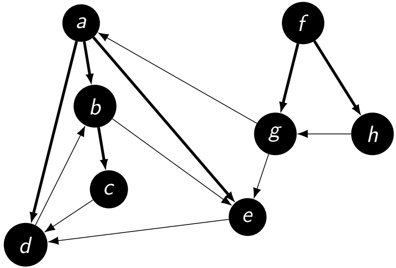
DFS vs. BFS example

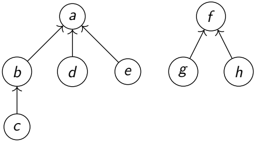
DFS:





BFS:





Agenda

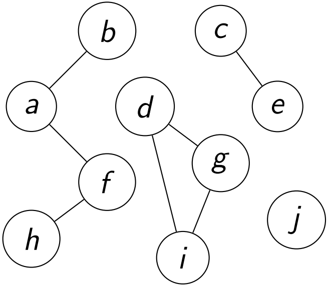
1. Depth First Search (DFS)
2. Connected components
   - Connected components
   - Strongly connected components

Connected components

- A undirected graph $G = (V, E)$ is **connected** if $\forall u, v \in V$ there is a path $u \rightsquigarrow v$ (u and v are reachable from one another).
- A **connected component** is a maximal set of vertices $C \subseteq V$ such that C is connected.

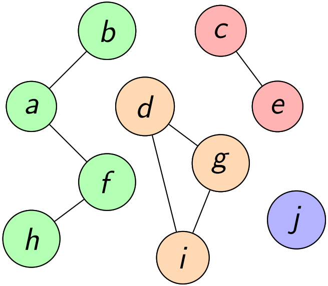
Connected components

- A undirected graph $G = (V, E)$ is **connected** if $\forall u, v \in V$ there is a path $u \rightsquigarrow v$ (u and v are reachable from one another).
- A **connected component** is a maximal set of vertices $C \subseteq V$ such that C is connected.



Connected components

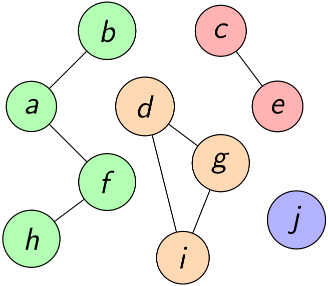
- A undirected graph $G = (V, E)$ is **connected** if $\forall u, v \in V$ there is a path $u \rightsquigarrow v$ (u and v are reachable from one another).
- A **connected component** is a maximal set of vertices $C \subseteq V$ such that C is connected.



- $\{a, b, f, h\}$, $\{c, e\}$, $\{d, g, i\}$ and $\{j\}$ are connected components

Connected components

- A undirected graph $G = (V, E)$ is **connected** if $\forall u, v \in V$ there is a path $u \rightsquigarrow v$ (u and v are reachable from one another).
- A **connected component** is a maximal set of vertices $C \subseteq V$ such that C is connected.



- $\{a, b, f, h\}$, $\{c, e\}$, $\{d, g, i\}$ and $\{j\}$ are connected components
- $\{a, b, c\}$ is not a connected component because $\nexists a \rightsquigarrow c$
- $\{a, b, h\}$ is not a connected component because it is not maximal

Finding connected components using DSF

CONNECTED-COMPONENTS(G)

for each vertex $v \in G.V$
    MAKE-SET(v)
for each edge $(u, v) \in G.E$
    if FIND-SET(u) $\neq$ FIND-SET(v)
        UNION(u, v)

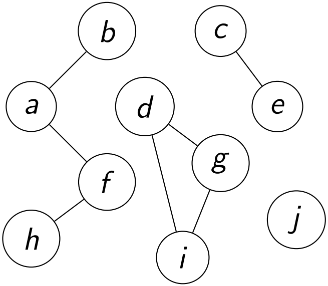
SAME-COMPONENT(u, v)

if FIND-SET(u) $==$ FIND-SET(v)
    return TRUE
else return FALSE

Finding connected components using DSF

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ G.V
    MAKE-SET(v)
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        UNION(u, v)
```

```
SAME-COMPONENT(u, v)
if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE
```



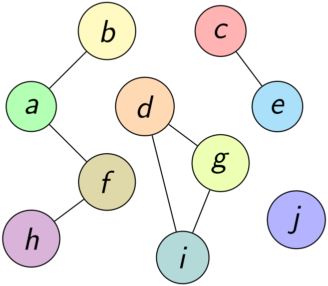
Finding connected components using DSF

CONNECTED-COMPONENTS(G)

for each vertex $v \in G.V$
  MAKE-SET(v)
for each edge $(u, v) \in G.E$
  if FIND-SET(u) $\neq$ FIND-SET(v)
    UNION(u, v)

SAME-COMPONENT(u, v)

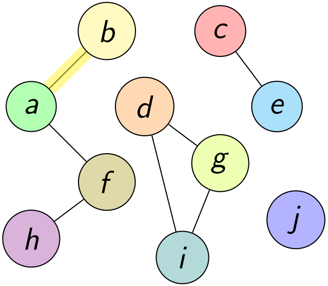
if FIND-SET(u) $==$ FIND-SET(v)
  return TRUE
else return FALSE



Finding connected components using DSF

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ G.V
    MAKE-SET(v)
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        UNION(u, v)
```

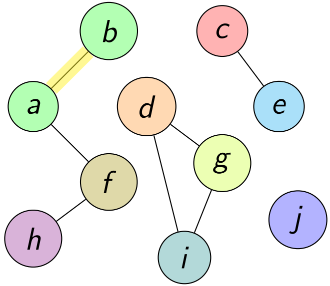
```
SAME-COMPONENT(u, v)
if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE
```



Finding connected components using DSF

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ G.V
    MAKE-SET(v)
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        UNION(u, v)
```

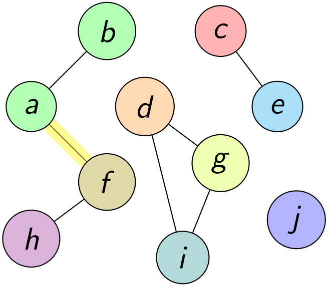
```
SAME-COMPONENT(u, v)
if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE
```



Finding connected components using DSF

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ G.V
    MAKE-SET(v)
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        UNION(u, v)
```

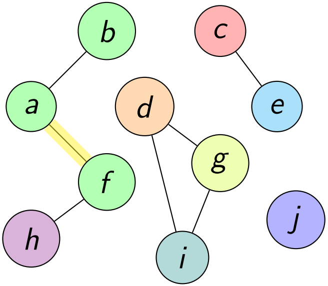
```
SAME-COMPONENT(u, v)
if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE
```



Finding connected components using DSF

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ G.V
    MAKE-SET(v)
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        UNION(u, v)
```

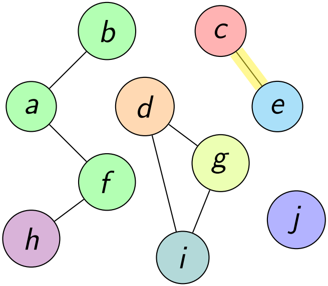
```
SAME-COMPONENT(u, v)
if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE
```



Finding connected components using DSF

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ G.V
    MAKE-SET(v)
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        UNION(u, v)
```

```
SAME-COMPONENT(u, v)
if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE
```



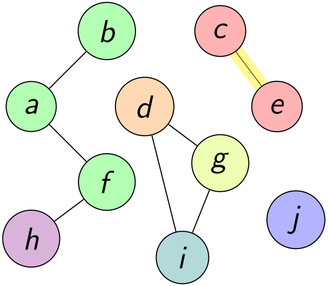
Finding connected components using DSF

CONNECTED-COMPONENTS(G)

for each vertex $v \in G.V$
  MAKE-SET(v)
for each edge $(u, v) \in G.E$
  if FIND-SET(u) $\neq$ FIND-SET(v)
    UNION(u, v)

SAME-COMPONENT(u, v)

if FIND-SET(u) $==$ FIND-SET(v)
  return TRUE
else return FALSE



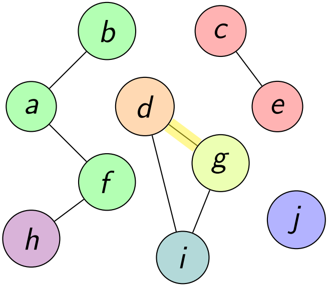
Finding connected components using DSF

CONNECTED-COMPONENTS(G)

1 **for** each vertex $v \in G.V$
2   MAKE-SET(v)
3 **for** each edge $(u, v) \in G.E$
4   **if** FIND-SET(u) $\neq$ FIND-SET(v)
5   UNION(u, v)

SAME-COMPONENT(u, v)

1 **if** FIND-SET(u) == FIND-SET(v)
2   **return** TRUE
3 **else return** FALSE



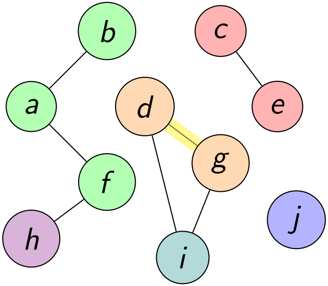
Finding connected components using DSF

CONNECTED-COMPONENTS(G)

for each vertex $v \in G.V$
    MAKE-SET(v)
for each edge $(u, v) \in G.E$
    if FIND-SET(u) $\neq$ FIND-SET(v)
        UNION(u, v)

SAME-COMPONENT(u, v)

if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE



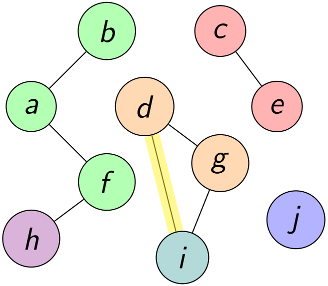
Finding connected components using DSF

Connected-Components(G)

```
for each vertex v ∈ G.V
    Make-Set(v)
for each edge (u, v) ∈ G.E
    if Find-Set(u) ≠ Find-Set(v)
        Union(u, v)
```

Same-Component(u, v)

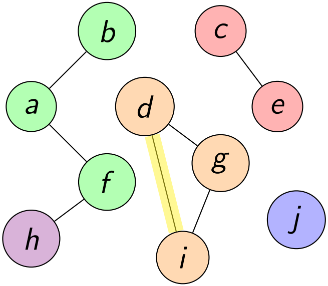
```
if Find-Set(u) == Find-Set(v)
    return TRUE
else return FALSE
```



Finding connected components using DSF

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ G.V
    MAKE-SET(v)
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        UNION(u, v)
```

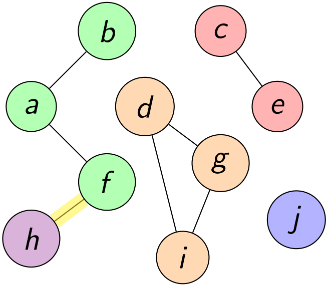
```
SAME-COMPONENT(u, v)
if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE
```



Finding connected components using DSF

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ G.V
    MAKE-SET(v)
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        UNION(u, v)
```

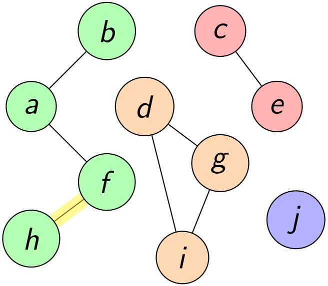
```
SAME-COMPONENT(u, v)
if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE
```



Finding connected components using DSF

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ G.V
    MAKE-SET(v)
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        UNION(u, v)
```

```
SAME-COMPONENT(u, v)
if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE
```



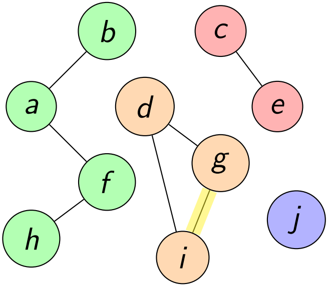
Finding connected components using DSF

CONNECTED-COMPONENTS(G)

```
for each vertex v ∈ G.V
    MAKE-SET(v)
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        UNION(u, v)
```

SAME-COMPONENT(u, v)

```
if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE
```



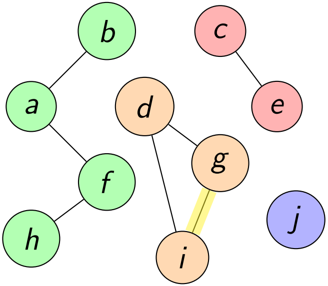
Finding connected components using DSF

CONNECTED-COMPONENTS(G)

for each vertex $v \in G.V$
    MAKE-SET(v)
for each edge $(u, v) \in G.E$
    if FIND-SET(u) $\neq$ FIND-SET(v)
        UNION(u, v)

SAME-COMPONENT(u, v)

if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE



Finding connected components using DFS

- use DFS as a skeleton

```
DFS-CC(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-CC(G, u)
```

```
DFS-VISIT-CC(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-CC(G, v)
u.color = BLACK

time = time + 1
u.f = time
```

Finding connected components using DFS

- use DFS as a skeleton
- add the *comp* attribute for each node
- the current *comp* value increases when a new DFS tree is started

```
DFS-CC(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
    u.comp = 0
time = 0
comp = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        comp = comp + 1
        DFS-VISIT-CC(G, u, comp)
```

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

Finding connected components using DFS

- use DFS as a skeleton
- add the *comp* attribute for each node
- the current *comp* value increases when a new DFS tree is started
- remove unwanted attributes like π, d, f

DFS-CC(G)

for each vertex $u \in G.V$
    $u.color = \text{WHITE}$
    $\cancel{u.\pi = \text{NIL}}$
    $u.comp = 0$
$\cancel{time = 0}$
$comp = 0$
for each vertex $u \in G.V$
    if $u.color == \text{WHITE}$
        $comp = comp + 1$
        DFS-VISIT-CC($G, u, comp$)

DFS-VISIT-CC($G, u, comp$)

$\cancel{time = time + 1}$
$\cancel{u.d = time}$
$u.color = \text{GRAY}$
for each $v \in G.Adj[u]$
    if $v.color == \text{WHITE}$
        $\cancel{v.\pi = u}$
        DFS-VISIT-CC($G, v, comp$)
$u.color = \text{BLACK}$
$u.comp = comp$
$\cancel{time = time + 1}$
$\cancel{u.f = time}$

Finding connected components using DFS

- use DFS as a skeleton
- add the *comp* attribute for each node
- the current *comp* value increases when a new DFS tree is started
- remove unwanted attributes like π, *d*, *f*
- the BFS algorithm can also be used instead of DFS

DFS-CC(G)

for each vertex $u \in G.V$
    $u.color = \text{WHITE}$
    ~~$u.\pi = \text{NIL}$~~
    $u.comp = 0$
~~$time = 0$~~
$comp = 0$
for each vertex $u \in G.V$
    if $u.color == \text{WHITE}$
        $comp = comp + 1$
        DFS-VISIT-CC($G, u, comp$)

DFS-VISIT-CC($G, u, comp$)

~~$time = time + 1$~~
~~$u.d = time$~~
$u.color = \text{GRAY}$
for each $v \in G.Adj[u]$
    if $v.color == \text{WHITE}$
        ~~$v.\pi = u$~~
        DFS-VISIT-CC($G, v, comp$)
$u.color = \text{BLACK}$
$u.comp = comp$
~~$time = time + 1$~~
~~$u.f = time$~~

Performance - DSF vs. DFS/BFS

- DSF approach
  - for each vertex we call MAKE-SET
  - for each edge we call FIND-SET and UNION

Performance - DSF vs. DFS/BFS

- DSF approach
  - for each vertex we call MAKE-SET
  - for each edge we call FIND-SET and UNION
  - $O(V + E \cdot \alpha(V))$

Performance - DSF vs. DFS/BFS

- DSF approach
  - for each vertex we call MAKE-SET
  - for each edge we call FIND-SET and UNION
  - $O(V + E \cdot \alpha(V))$
- DFS/BFS

Performance - DSF vs. DFS/BFS

- DSF approach
    - for each vertex we call MAKE-SET
    - for each edge we call FIND-SET and UNION
    - $O(V + E \cdot \alpha(V))$
- DFS/BFS
    - $O(V + E)$

Performance - DSF vs. DFS/BFS

- DSF approach
  - for each vertex we call MAKE-SET
  - for each edge we call FIND-SET and UNION
  - $O(V + E \cdot \alpha(V))$
- DFS/BFS
  - $O(V + E)$
- normally, the DFS/BFS approach is faster

Performance - DSF vs. DFS/BFS

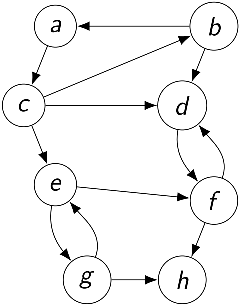
- DSF approach
  - for each vertex we call MAKE-SET
  - for each edge we call FIND-SET and UNION
  - $O(V + E \cdot \alpha(V))$
- DFS/BFS
  - $O(V + E)$
- normally, the DFS/BFS approach is faster
- there are custom scenarios where the DSF approach is preferred

Strongly connected components

- A directed graph $G = (V, E)$ is **strongly connected** if $\forall u, v \in V$ there is a path $u \rightsquigarrow v$ and a path $v \rightsquigarrow u$ (u and v are reachable from one another).
- A **strongly connected component (SCC)** is a maximal set of vertices $C \subseteq V$ such that C is strongly connected.

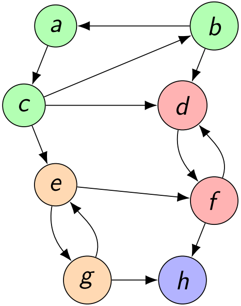
Strongly connected components

- A directed graph $G = (V, E)$ is **strongly connected** if $\forall u, v \in V$ there is a path $u \leadsto v$ and a path $v \leadsto u$ (u and v are reachable from one another).
- A **strongly connected component (SCC)** is a maximal set of vertices $C \subseteq V$ such that C is strongly connected.



Strongly connected components

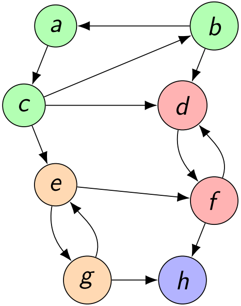
- A directed graph $G = (V, E)$ is **strongly connected** if $\forall u, v \in V$ there is a path $u \rightsquigarrow v$ and a path $v \rightsquigarrow u$ (u and v are reachable from one another).
- A **strongly connected component (SCC)** is a maximal set of vertices $C \subseteq V$ such that C is strongly connected.



- $\{a, b, c\}$, $\{d, f\}$, $\{e, g\}$ and $\{h\}$ are SCC

Strongly connected components

- A directed graph $G = (V, E)$ is **strongly connected** if $\forall u, v \in V$ there is a path $u \rightsquigarrow v$ and a path $v \rightsquigarrow u$ (u and v are reachable from one another).
- A **strongly connected component (SCC)** is a maximal set of vertices $C \subseteq V$ such that C is strongly connected.



- $\{a, b, c\}$, $\{d, f\}$, $\{e, g\}$ and $\{h\}$ are SCC
- $\{a, b, c, d\}$ is not an SCC because $\nexists d \rightsquigarrow a$ (although $\exists a \rightsquigarrow d$)

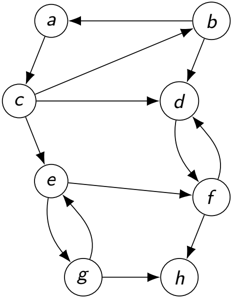
SCC algorithm (Kosaraju)

```
STRONGLY-CONNECTED-COMPONENTS(G)
1   call DFS(G) to compute finishing times u.f, for each u ∈ G.V
2   compute G^T by reversing each edge direction
3   call DFS(G^T), but in the main loop consider the vertices
    in order of decreasing u.f (as computed in line 1)
4   output the vertices of each tree in the DFS forest (line 3) as a separate SCC
```

SCC algorithm (Kosaraju)

STRONGLY-CONNECTED-COMPONENTS(G)

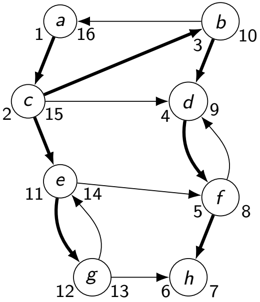
1 call DFS(G) to compute finishing times $u.f$, for each $u \in G.V$
2 compute G^T by reversing each edge direction
3 call DFS(G^T), but in the main loop consider the vertices
  in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the DFS forest (line 3) as a separate SCC



SCC algorithm (Kosaraju)

STRONGLY-CONNECTED-COMPONENTS(G)

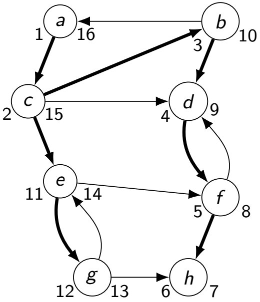
1 call DFS(G) to compute finishing times $u.f$, for each $u \in G.V$
2 compute G^T by reversing each edge direction
3 call DFS(G^T), but in the main loop consider the vertices
  in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the DFS forest (line 3) as a separate SCC

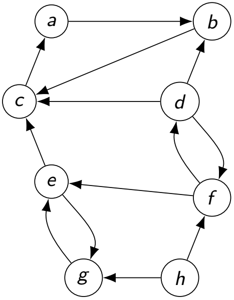


SCC algorithm (Kosaraju)

STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finishing times $u.f$, for each $u \in G.V$
2 compute G^T by reversing each edge direction
3 call DFS(G^T), but in the main loop consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the DFS forest (line 3) as a separate SCC

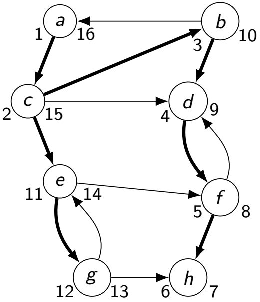


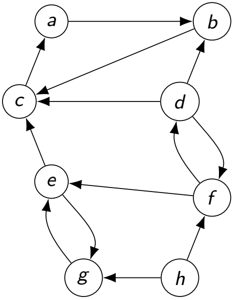


SCC algorithm (Kosaraju)

STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finishing times $u.f$, for each $u \in G.V$
2 compute G^T by reversing each edge direction
3 call DFS(G^T), but in the main loop consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the DFS forest (line 3) as a separate SCC



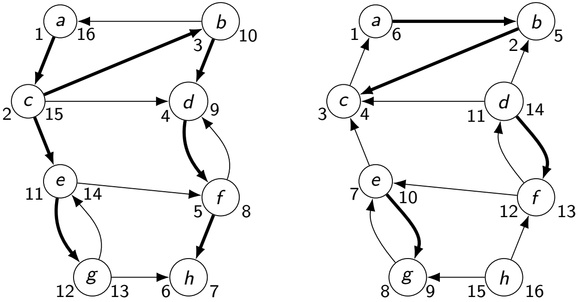


vertex order:
a, c, e, g, b, d, f, h

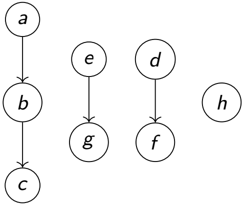
SCC algorithm (Kosaraju)

STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finishing times $u.f$, for each $u \in G.V$
2 compute G^T by reversing each edge direction
3 call DFS(G^T), but in the main loop consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the DFS forest (line 3) as a separate SCC



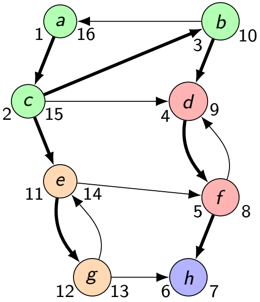
vertex order:
a, c, e, g, b, d, f, h

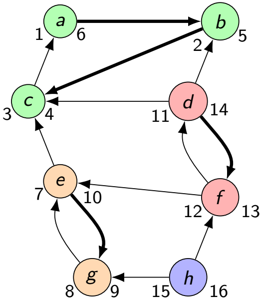


SCC algorithm (Kosaraju)

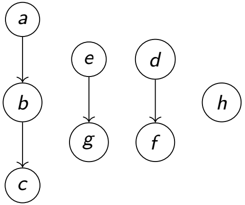
STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finishing times $u.f$, for each $u \in G.V$
2 compute G^T by reversing each edge direction
3 call DFS(G^T), but in the main loop consider the vertices
in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the DFS forest (line 3) as a separate SCC





vertex order:
a, c, e, g, b, d, f, h



SCC algorithm - Complexity

STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finishing times $u.f$, for each $u \in G.V$
2 compute G^T by reversing each edge direction
3 call DFS(G^T), but in the main loop consider the vertices
  in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the DFS forest (line 3) as a separate SCC

SCC algorithm - Complexity

STRONGLY-CONNECTED-COMPONENTS(G)

1 call $\mathrm{DFS}(G)$ to compute finishing times $u.f$, for each $u \in G.V$
2 compute G^T by reversing each edge direction
3 call $\mathrm{DFS}(G^T)$, but in the main loop consider the vertices
  in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the DFS forest (line 3) as a separate SCC

1. DFS complexity: $O(V + E)$

SCC algorithm - Complexity

STRONGLY-CONNECTED-COMPONENTS(G)

1 call $\text{DFS}(G)$ to compute finishing times $u.f$, for each $u \in G.V$
2 compute G^T by reversing each edge direction
3 call $\text{DFS}(G^T)$, but in the main loop consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the DFS forest (line 3) as a separate SCC

1. DFS complexity: $O(V + E)$
2. computing G^T: $O(E)$

SCC algorithm - Complexity

STRONGLY-CONNECTED-COMPONENTS(G)

1. call $\mathrm{DFS}(G)$ to compute finishing times $u.f$, for each $u \in G.V$
2. compute G^T by reversing each edge direction
3. call $\mathrm{DFS}(G^T)$, but in the main loop consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4. output the vertices of each tree in the DFS forest (line 3) as a separate SCC

1. DFS complexity: $O(V + E)$
2. computing G^T: $O(E)$
3. DFS complexity: $O(V + E)$

SCC algorithm - Complexity

STRONGLY-CONNECTED-COMPONENTS(G)

```
1  call DFS(G) to compute finishing times u.f, for each u ∈ G.V
2  compute G^T by reversing each edge direction
3  call DFS(G^T), but in the main loop consider the vertices
   in order of decreasing u.f (as computed in line 1)
4  output the vertices of each tree in the DFS forest (line 3) as a separate SCC
```

1. DFS complexity: $O(V+E)$
2. computing G^T: $O(E)$
3. DFS complexity: $O(V+E)$
4. traversing the DFS forest: $O(V)$

SCC algorithm - Complexity

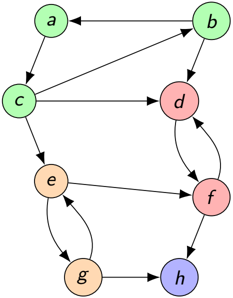
```
STRONGLY-CONNECTED-COMPONENTS(G)
1   call DFS(G) to compute finishing times u.f, for each u ∈ G.V
2   compute G^T by reversing each edge direction
3   call DFS(G^T), but in the main loop consider the vertices
    in order of decreasing u.f (as computed in line 1)
4   output the vertices of each tree in the DFS forest (line 3) as a separate SCC
```

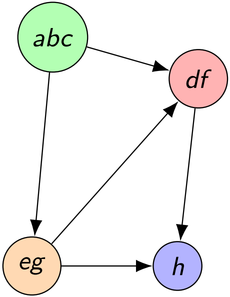
1. DFS complexity: $O(V + E)$
2. computing G^T: $O(E)$
3. DFS complexity: $O(V + E)$
4. traversing the DFS forest: $O(V)$

The algorithm complexity is $O(V + E)$.

SCC algorithm - Proof (1)

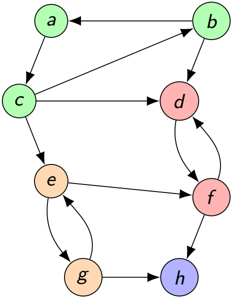
- the component graph $G^{SCC} = (V^{SCC}, E^{SCC})$
- the SCC are $C_1, C_2, \ldots C_k$
- $V^{SCC} = \{v_1, v_2, \ldots v_k\}$
- $(v_i, v_j) \in E^{SCC}$ if $\exists (x, y) \in E$, such that $x \in C_i$, $y \in C_j$

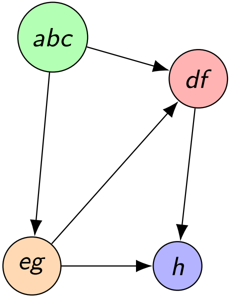




SCC algorithm - Proof (1)

- the component graph $G^{SCC} = (V^{SCC}, E^{SCC})$
- the SCC are $C_1, C_2, \dots C_k$
- $V^{SCC} = \{v_1, v_2, \dots v_k\}$
- $(v_i, v_j) \in E^{SCC}$ if $\exists (x, y) \in E$, such that $x \in C_i$, $y \in C_j$

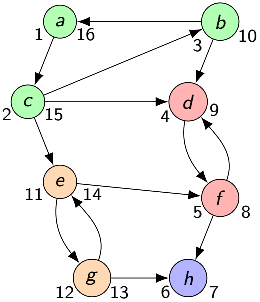


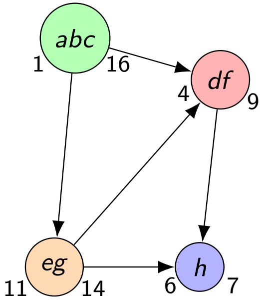


- G and G^T have the same SCC

SCC algorithm - Proof (1)

- the component graph $G^{SCC} = (V^{SCC}, E^{SCC})$
- the SCC are $C_1, C_2, \dots C_k$
- $V^{SCC} = \{v_1, v_2, \dots v_k\}$
- $(v_i, v_j) \in E^{SCC}$ if $\exists (x, y) \in E$, such that $x \in C_i$, $y \in C_j$





- G and G^T have the same SCC
- let C be a SCC of G
- $d(C) = \min_{u \in C}\{u.d\}$
- $f(C) = \max_{u \in C}\{u.f\}$

SCC algorithm - Proof (2)

Lemma 22.13

Let C and C' be distinct SCC in directed graph G. Let $u, v \in C$ and $u', v' \in C'$ and suppose G contains a path $u \rightsquigarrow u'$. Then G cannot contain a path $v' \rightsquigarrow v$.

SCC algorithm - Proof (2)

Lemma 22.13

Let C and C' be distinct SCC in directed graph G. Let $u, v \in C$ and $u', v' \in C'$ and suppose G contains a path $u \rightsquigarrow u'$. Then G cannot contain a path $v' \rightsquigarrow v$.

- by contradiction, if $\exists v' \rightsquigarrow v$, then
  - $\exists u \rightsquigarrow u' \rightsquigarrow v'$
  - $\exists v' \rightsquigarrow v \rightsquigarrow u$
- $\Rightarrow$ u and v' are reachable from each other
- $\Rightarrow$ C and C' are not distinct

SCC algorithm - Proof (3)

Lemma 22.14

Let C and C' be distinct SCC in directed graph $G = (V, E)$. Suppose that $(u, v) \in E$, with $u \in C$ and $v \in C'$. Then $f(C) > f(C')$.

SCC algorithm - Proof (3)

Lemma 22.14

Let C and C' be distinct SCC in directed graph $G = (V, E)$. Suppose that $(u, v) \in E$, with $u \in C$ and $v \in C'$. Then $f(C) > f(C')$.

- if $d(C) < d(C')$
  - Let x be the first vertex discovered in C. At the time $x.d$ all the vertices in C and C' are WHITE.
  - Any vertex in C and C' is a descendant of x in the DFS tree ($x \rightsquigarrow u \rightarrow v \rightsquigarrow y, \forall y \in C'$).
  - $\Rightarrow x.f = f(C) > f(C')$

SCC algorithm - Proof (3)

Lemma 22.14

Let C and C' be distinct SCC in directed graph $G = (V, E)$. Suppose that $(u, v) \in E$, with $u \in C$ and $v \in C'$. Then $f(C) > f(C')$.

- if $d(C) < d(C')$
  - Let x be the first vertex discovered in C. At the time $x.d$ all the vertices in C and C' are WHITE.
  - Any vertex in C and C' is a descendant of x in the DFS tree ($x \rightsquigarrow u \rightarrow v \rightsquigarrow y, \forall y \in C'$).
  - $\Rightarrow x.f = f(C) > f(C')$
- if $d(C) > d(C')$
  - Let y be the first vertex discovered in C'. At the time $y.d$, all the vertices in C and C' are WHITE.
  - Any vertex in C' is a descendant of y.
  - From *Lemma 22.13*, $\nexists y \rightsquigarrow x, \forall x \in C$.
  - At the time $y.f$, C is still WHITE, so $f(C) > y.f = f(C')$.

SCC algorithm - Proof (4)

Corollary 22.15

Let C and C' be distinct SCC in directed graph $G = (V, E)$.
Suppose there is an edge $(u, v) \in E^T$ where $u \in C$ and $v \in C'$.
Then $f(C) < f(C')$.

SCC algorithm - Proof (4)

Corollary 22.15

Let C and C' be distinct SCC in directed graph $G = (V, E)$.
Suppose there is an edge $(u, v) \in E^T$ where $u \in C$ and $v \in C'$.
Then $f(C) < f(C')$.

- Since $(u, v) \in E^T$, we have $(v, u) \in E$.
- Since G and G^T have the same SCC, from *Lemma 22.14* $\Rightarrow f(C) < f(C')$.

SCC algorithm - Proof (5)

Theorem

The STRONGLY-CONNECTED-COMPONENTS procedure correctly computes the SCC of the directed graph G provided as input.

SCC algorithm - Proof (5)

Theorem

The STRONGLY-CONNECTED-COMPONENTS procedure correctly computes the SCC of the directed graph G provided as input.

- proof by induction on the number of DFS trees produced by calling DFS on G^T
- The induction hypothesis is that the trees $T_1, T_2, \dots T_k$ are SCC and we need to prove that T_{k+1} is also a SCC. $k = 0$ is trivial.
- Let the root of T_{k+1} be u, that belongs to the SCC C.
- Since the DFS on G^T considers the vertices in descending order by finishing time, $u.f = f(C) > f(C')$ for any SCC C' that has yet to be visited.
- By the inductive hypothesis, at the time $u.d$, all nodes $v \in C$ are WHITE so they will be descendants of u in T_{k+1}. $\Rightarrow C \subseteq T_{k+1}$.
- From *Corollary 22.15*, any SCC C' that is reachable from C (in G^T) has $f(C) < f(C')$ so it has already been visited. $\Rightarrow T_{k+1} \subseteq C$.
- $C \subseteq T_{k+1} \wedge T_{k+1} \subseteq C \rightarrow T_{k+1} = C$, so T_{k+1} is an SCC.

Bibliography

- Cormen, Thomas H., et al., *"Introduction to algorithms."*, MIT press, 2009, cap. 22.3, 22.4, 22.5