Graph Algorithms II

Depth First Search

Depth First Search (DFS)

Depth First Search (DFS)

- BFS : First Discovered – First Explored

Depth First Search (DFS)

- BFS : First Discovered – First Explored
- DFS: Last Discovered – First Explored

Depth First Search (DFS)

- BFS : First Discovered – First Explored
- DFS: Last Discovered – First Explored

The algorithm keeps track of

Depth First Search (DFS)

- BFS : First Discovered – First Explored
- DFS: Last Discovered – First Explored

The algorithm keeps track of

color[v] = white, grey, or black

Depth First Search (DFS)

- BFS : First Discovered – First Explored
- DFS: Last Discovered – First Explored

The algorithm keeps track of

color[v] = white, grey, or black

p[v] = u iff u discovered v

Depth First Search (DFS)

- BFS : First Discovered – First Explored
- DFS: Last Discovered – First Explored

The algorithm keeps track of

Same as BFS

color[v] = white, grey, or black

p[v] = u iff u discovered v

Depth First Search (DFS)

- BFS : First Discovered – First Explored
- DFS: Last Discovered – First Explored

The algorithm keeps track of

Same as BFS

color[v] = white, grey, or black

p[v] = u iff u discovered v

Different from BFS

Depth First Search (DFS)

- BFS : First Discovered – First Explored
- DFS: Last Discovered – First Explored

The algorithm keeps track of

Same as BFS

color[v] = white, grey, or black

p[v] = u iff u discovered v

Different from BFS

d[v] = “time” when v is discovered

Depth First Search (DFS)

- BFS : First Discovered – First Explored
- DFS: Last Discovered – First Explored

The algorithm keeps track of

Same as BFS

color[v] = white, grey, or black

p[v] = u iff u discovered v

Different from BFS

d[v] = "time" when v is discovered

f[v] = "time" when v's exploration is completed

Depth First Search (DFS)

- BFS : First Discovered – First Explored
- DFS: Last Discovered – First Explored

The algorithm keeps track of

Same as BFS

color[v] = white, grey, or black

p[v] = u iff u discovered v

Different from BFS

d[v] = "time" when v is discovered

f[v] = "time" when v's exploration is completed

To keep track of "time", it uses a counter which is incremented by 1

- whenever a new node is discovered
- whenever it finishes exploring a node

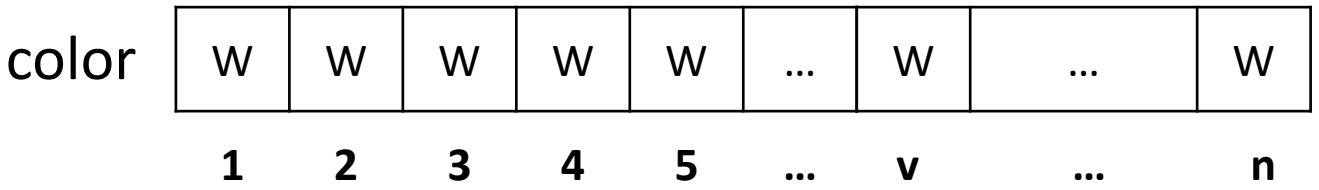
DFS(G**)** /* G = (V, E) */

DFS(G**)** /* G = (V, E) */

For each v ∈ V **do**

color[v] ← white ;

End For

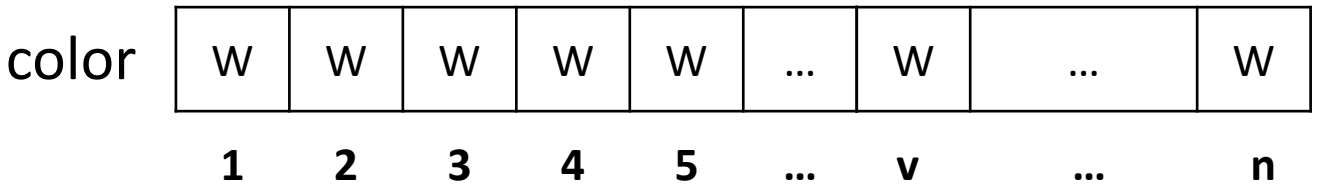


DFS(G**)** /* G = (V, E) */

For each v ∈ V **do**

color[v] ← white ; d[v] ← ∞ ;

End For

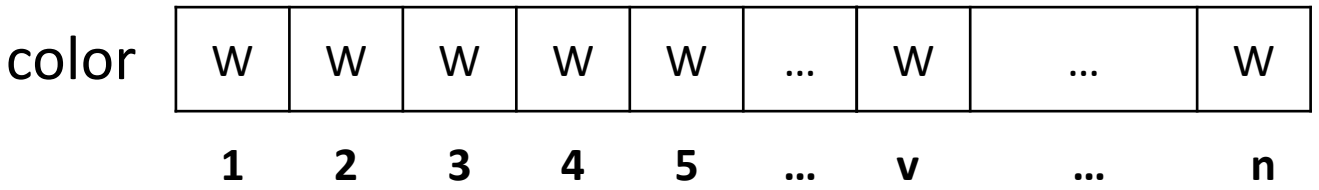


DFS(G**)** /* G = (V, E) */

For each v ∈ V **do**

color[v] ← white ; d[v] ← ∞ ; f[v] ← ∞ ;

End For

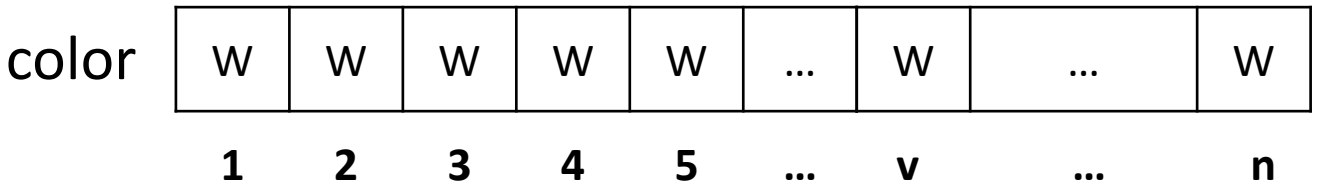


DFS(G**)** /* G = (V, E) */

For each v ∈ V **do**

color[v] ← white ; d[v] ← ∞ ; f[v] ← ∞ ; p[v] ← NIL

End For



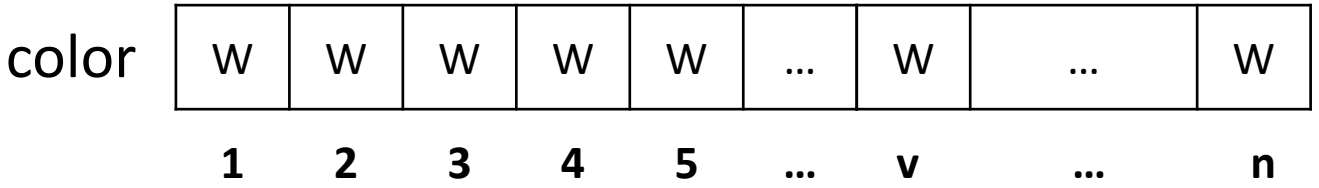
DFS(G**)** /* G = (V, E) */

For each v ∈ V **do**

color[v] ← white ; d[v] ← ∞ ; f[v] ← ∞ ; p[v] ← NIL

End For

time ← 0 /* Global variable */



DFS(G**)** /* G = (V, E) */

For each v ∈ V **do**

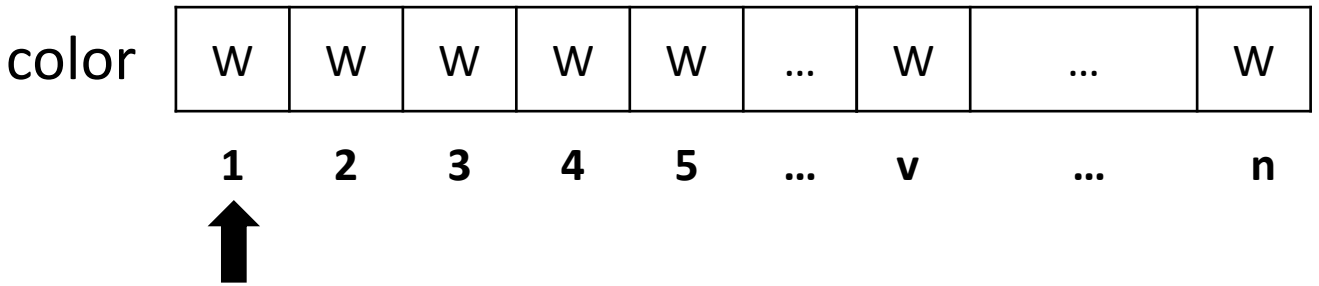
color[v] ← white ; d[v] ← ∞ ; f[v] ← ∞ ; p[v] ← NIL

End For

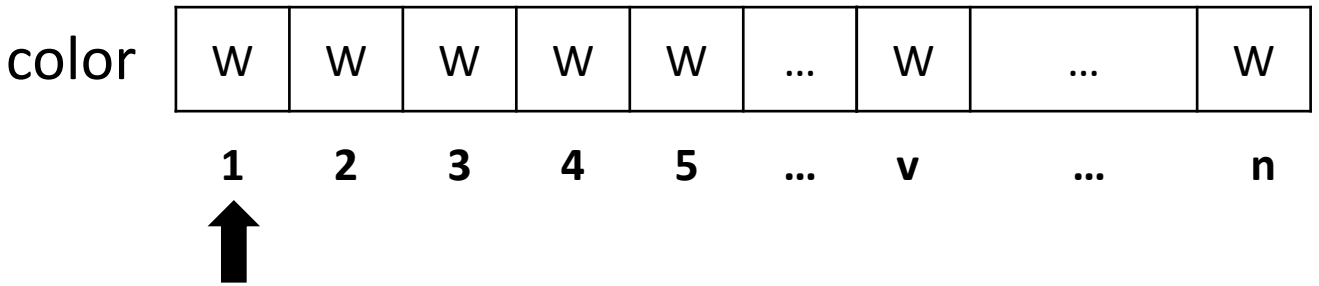
time ← 0 /* Global variable */

For each v ∈ V **do**

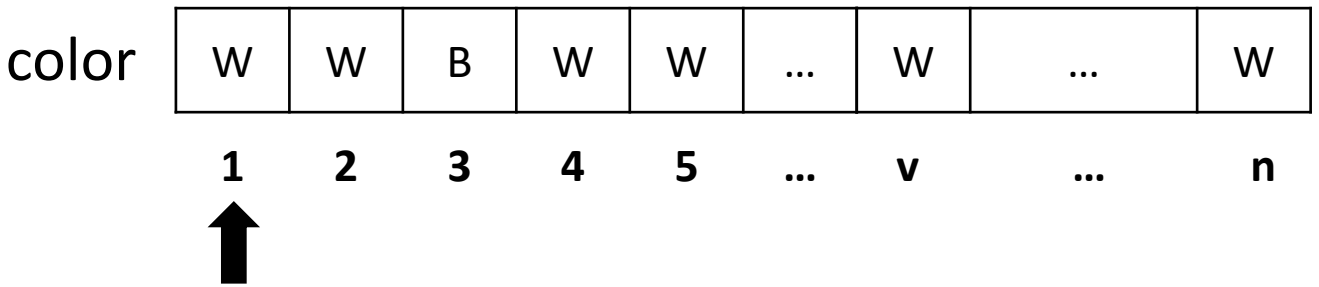
If color[v] = white



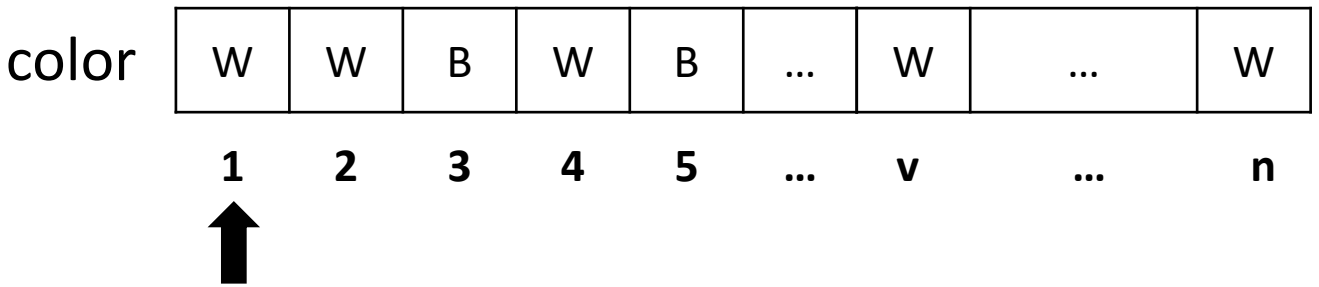
```
DFS(G)                          /*   G = (V, E)    */
    For each v ∈ V do
        color[v]  ← white ;  d[v] ← ∞ ;  f[v] ← ∞ ;  p[v] ← NIL
    End For
    time ← 0                    /*   Global variable   */
    For each v ∈ V do
        If color[v] = white  then DFS-Explore(G, v)
    End For
End DFS
```



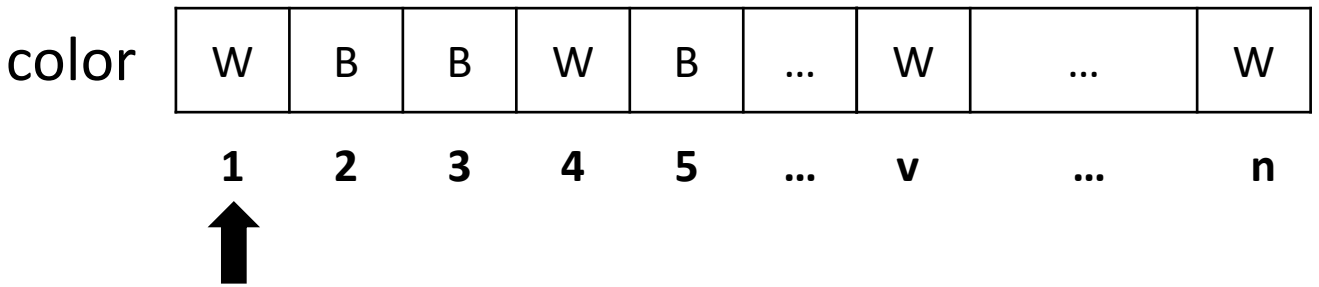
```
DFS(G)                          /*   G = (V, E)    */
    For each v ∈ V do
        color[v]  ← white ;  d[v] ← ∞ ;  f[v] ← ∞ ;  p[v] ← NIL
    End For
    time ← 0                    /*   Global variable   */
    For each v ∈ V do
        If color[v] = white  then DFS-Explore(G, v)
    End For
End DFS
```



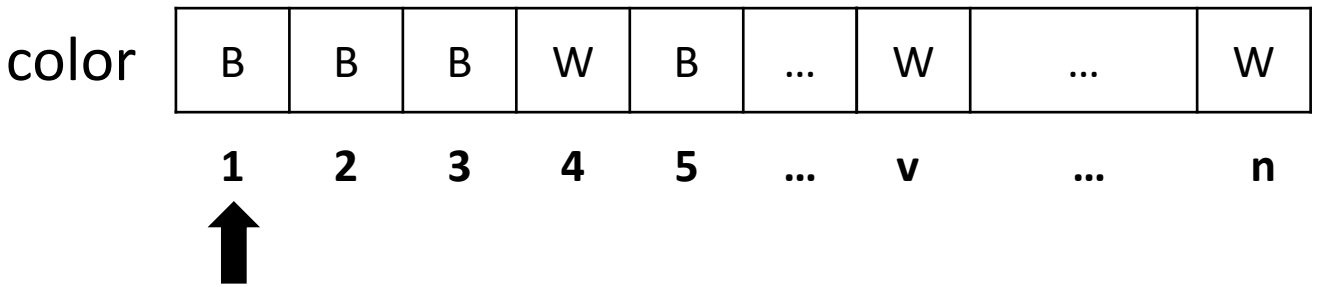
```
DFS(G)                          /*   G = (V, E)    */
    For each v ∈ V do
        color[v]  ← white ;  d[v] ← ∞ ;  f[v] ← ∞ ;  p[v] ← NIL
    End For
    time ← 0                    /*   Global variable   */
    For each v ∈ V do
        If color[v] = white  then DFS-Explore(G, v)
    End For
End DFS
```



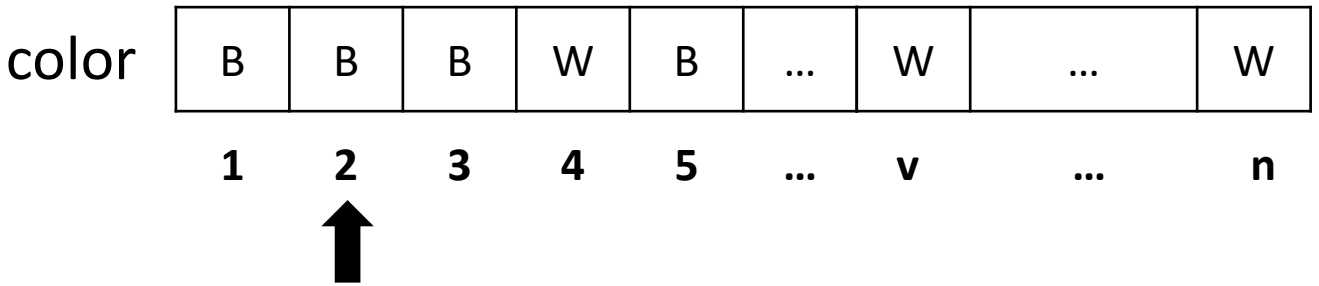
```
DFS(G)                          /*   G = (V, E)    */
    For each v ∈ V do
        color[v]  ← white ;  d[v] ← ∞ ;  f[v] ← ∞ ;  p[v] ← NIL
    End For
    time ← 0                    /*   Global variable   */
    For each v ∈ V do
        If color[v] = white  then DFS-Explore(G, v)
    End For
End DFS
```



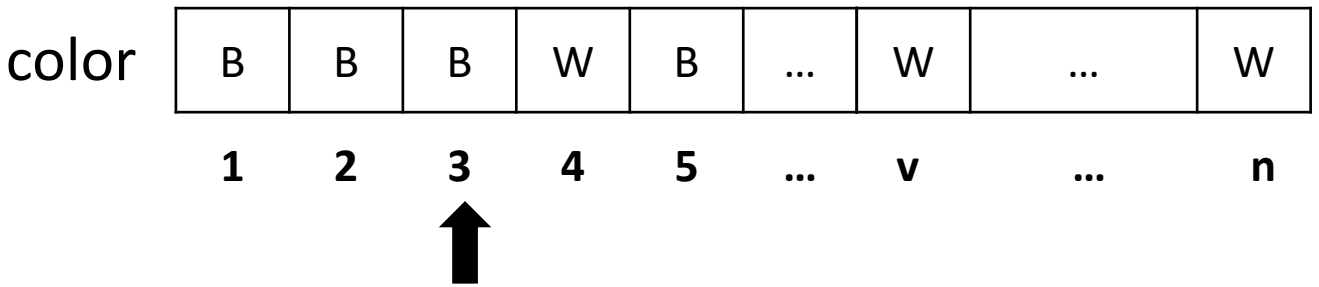
```
DFS(G)                          /*   G = (V, E)    */
    For each v ∈ V do
        color[v]  ← white ;  d[v] ← ∞ ;  f[v] ← ∞ ;  p[v] ← NIL
    End For
    time ← 0                    /*   Global variable   */
    For each v ∈ V do
        If color[v] = white  then DFS-Explore(G, v)
    End For
End DFS
```



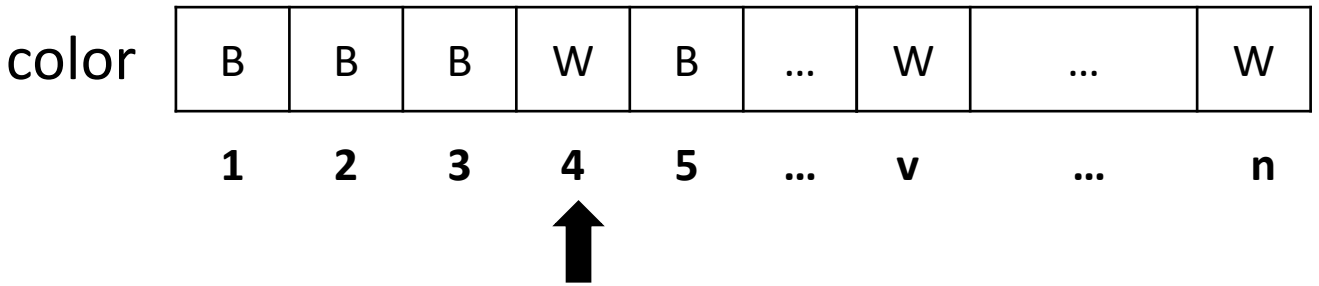
```
DFS(G)                          /*   G = (V, E)    */
    For each v ∈ V do
        color[v]  ← white ;  d[v] ← ∞ ;  f[v] ← ∞ ;  p[v] ← NIL
    End For
    time ← 0                    /*   Global variable   */
    For each v ∈ V do
        If color[v] = white  then DFS-Explore(G, v)
    End For
End DFS
```



```
DFS(G)                          /*   G = (V, E)    */
    For each v ∈ V do
        color[v]  ← white ;  d[v] ← ∞ ;  f[v] ← ∞ ;  p[v] ← NIL
    End For
    time ← 0                    /*   Global variable   */
    For each v ∈ V do
        If color[v] = white  then DFS-Explore(G, v)
    End For
End DFS
```



```
DFS(G)                          /*   G = (V, E)    */
    For each v ∈ V do
        color[v]  ← white ;  d[v] ← ∞ ;  f[v] ← ∞ ;  p[v] ← NIL
    End For
    time ← 0                    /*   Global variable   */
    For each v ∈ V do
        If color[v] = white  then DFS-Explore(G, v)
    End For
End DFS
```



DFS-Explore(G, u**)** /* Explore node u */

```
DFS-Explore(G, u)                    /*  Explore node u                */
    color[u]  ← grey                 /*  u was just discovered         */
```


```
DFS-Explore(G, u)                          /*  Explore node u                */
    color[u]  ← grey                       /*  u was just discovered         */
    time  ← time + 1  ;  d[u]  ← time      /*   u's discovery time           */
```


```
DFS-Explore(G, u)                          /*  Explore node u                */
    color[u]  ← grey                       /*  u was just discovered         */
    time  ← time + 1  ;  d[u]  ← time      /*   u's discovery time           */
    For each edge (u, v) ∈ E do            /*   Explore edge (u,v)           */
```


```
DFS-Explore(G, u)                                  /*  Explore node u                */
    color[u]  ← grey                               /*  u was just discovered         */
    time  ← time + 1  ;  d[u]  ← time              /*   u's discovery time           */
    For each edge (u, v) ∈ E do                    /*   Explore edge (u,v)           */
        If color[v] = white then do                /*   u discovered v               */
```


```
DFS-Explore(G, u)                          /*  Explore node u                */
    color[u]  ← grey                       /*  u was just discovered         */
    time  ← time + 1  ;  d[u]  ← time      /*   u's discovery time           */
    For each edge (u, v) ∈ E do            /*   Explore edge (u,v)           */
        If color[v] = white then do        /*   u discovered v               */
            p[v]  ←   u
```


```
DFS-Explore(G, u)                          /*  Explore node u                */
    color[u]  ← grey                       /*  u was just discovered         */
    time  ← time + 1  ;  d[u]  ← time       /*   u's discovery time           */
    For each edge (u, v) ∈ E do            /*   Explore edge (u,v)           */
        If color[v] = white then do        /*   u discovered v               */
            p[v]  ←   u
            DFS-Explore(G, v)              /*   Explore v                    */
```

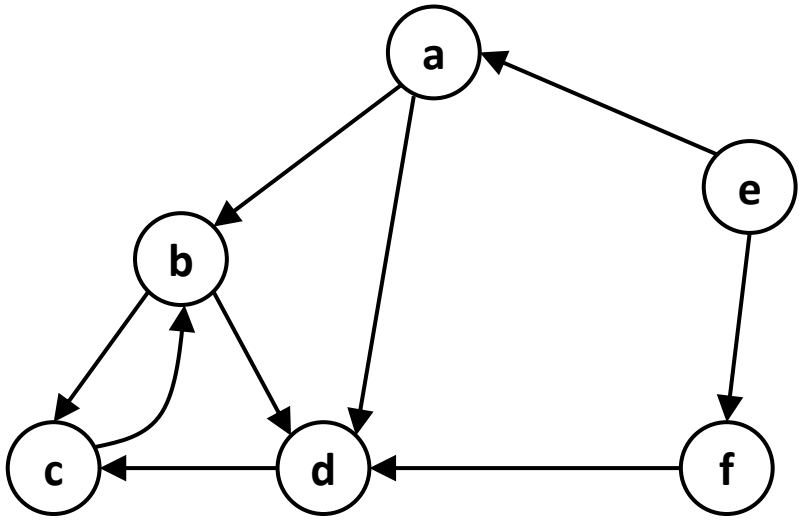

```
DFS-Explore(G, u)                          /*  Explore node u                */
    color[u]  ← grey                       /*  u was just discovered         */
    time  ← time + 1  ;  d[u]  ← time       /*   u's discovery time           */
    For each edge (u, v) ∈ E do            /*   Explore edge (u,v)           */
        If color[v] = white then do        /*   u discovered v               */
                p[v]  ←   u
                DFS-Explore(G, v)          /*   Explore v                    */
        End If
    End For
```


```
DFS-Explore(G, u)                          /*  Explore node u                  */
    color[u]  ← grey                       /*  u was just discovered           */
    time  ← time + 1  ;  d[u]  ← time      /*   u's discovery time             */
    For each edge (u, v) ∈ E do            /*   Explore edge (u,v)             */
        If color[v] = white then do        /*   u discovered v                 */
                p[v]  ←   u
                DFS-Explore(G, v)          /*   Explore v                      */
        End If
    End For
    color[u]  ←   black                    /* u's exploration is over          */
```

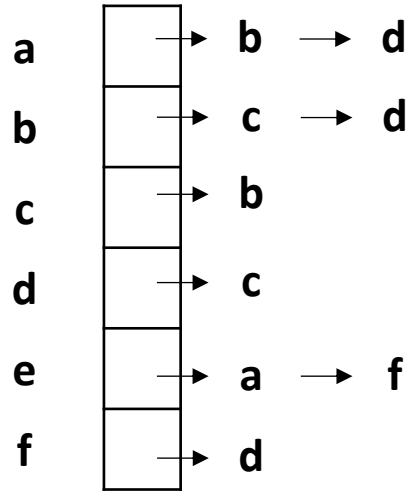
```
DFS-Explore(G, u)                          /*  Explore node u                  */
    color[u]  ← grey                       /*  u was just discovered           */
    time  ← time + 1  ;  d[u]  ← time      /*   u's discovery time             */
    For each edge (u, v) ∈ E do            /*   Explore edge (u,v)             */
        If color[v] = white then do        /*   u discovered v                 */
                p[v]  ←   u
                DFS-Explore(G, v)          /*   Explore v                      */
        End If
    End For
    color[u]  ←   black                    /* u's exploration is over          */
    time  ← time + 1  ;  f[u]  ← time      /* time u's exploration is over     */
```

```
DFS-Explore(G, u)                    /*  Explore node u                  */
    color[u]  ← grey                 /*  u was just discovered           */
    time  ← time + 1  ;  d[u]  ← time /*   u's discovery time             */
    For each edge (u, v) ∈ E do      /*   Explore edge (u,v)             */
        If color[v] = white then do  /*   u discovered v                 */
            p[v]  ←  u
            DFS-Explore(G, v)        /*   Explore v                      */
        End If
    End For
    color[u]  ←  black               /* u's exploration is over         */
    time  ← time + 1  ;  f[u]  ← time /* time u's exploration is over */
End DFS-Explore
```

DFS(G)



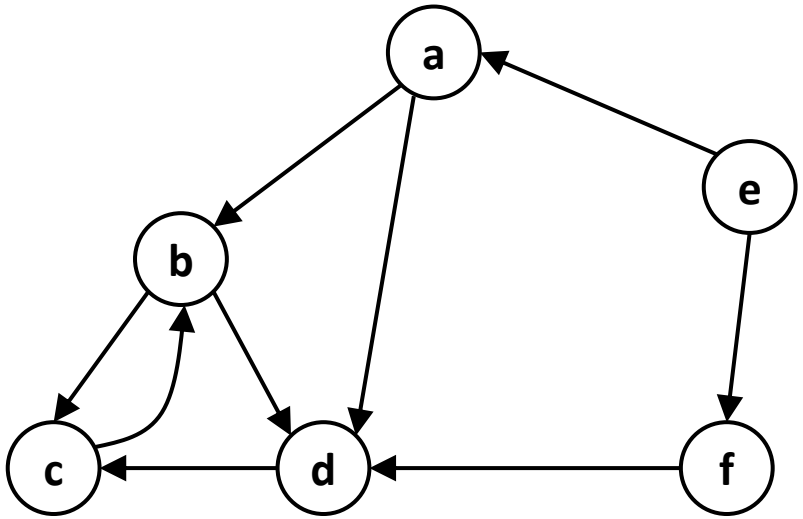
Adj List of G :



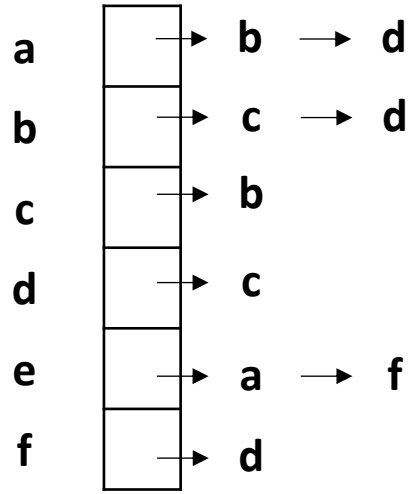
Colors of nodes:

a	b	c	d	e	f
W	W	W	W	W	W

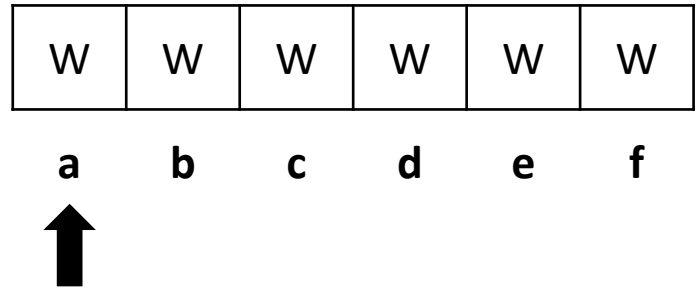
DFS(G)



Adj List of G :

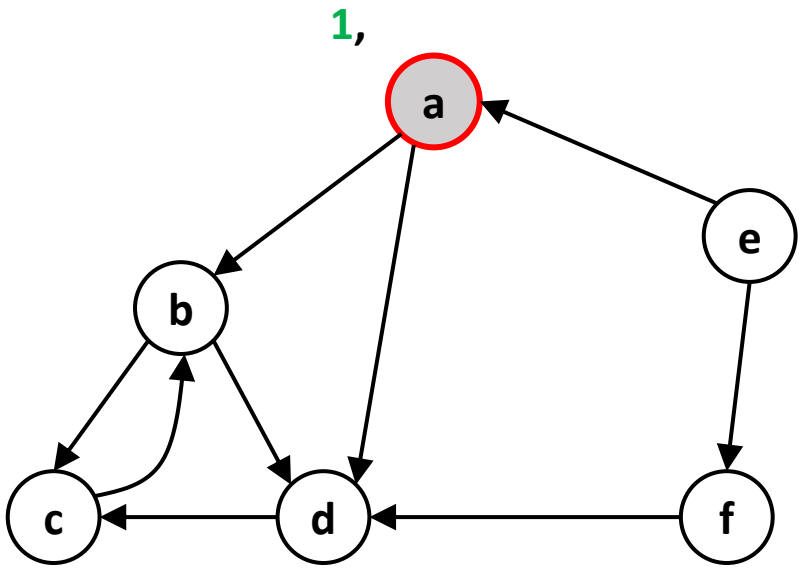


Colors of nodes:

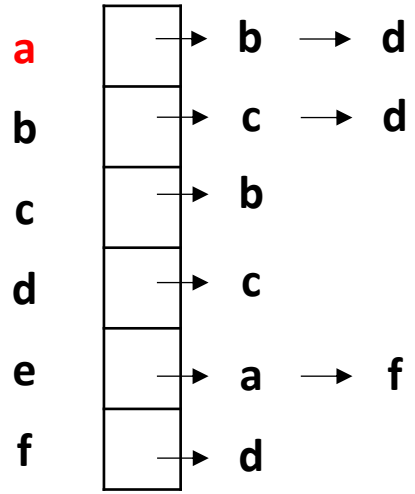


DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

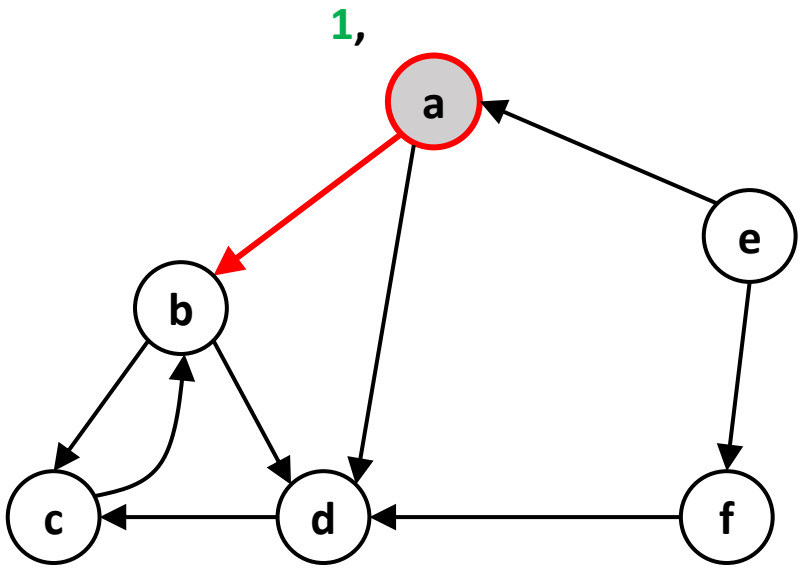


Colors of nodes:

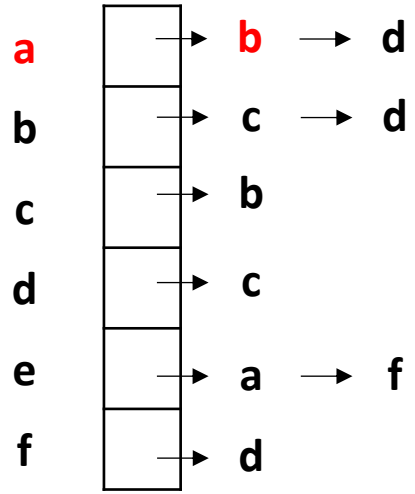
G	W	W	W	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

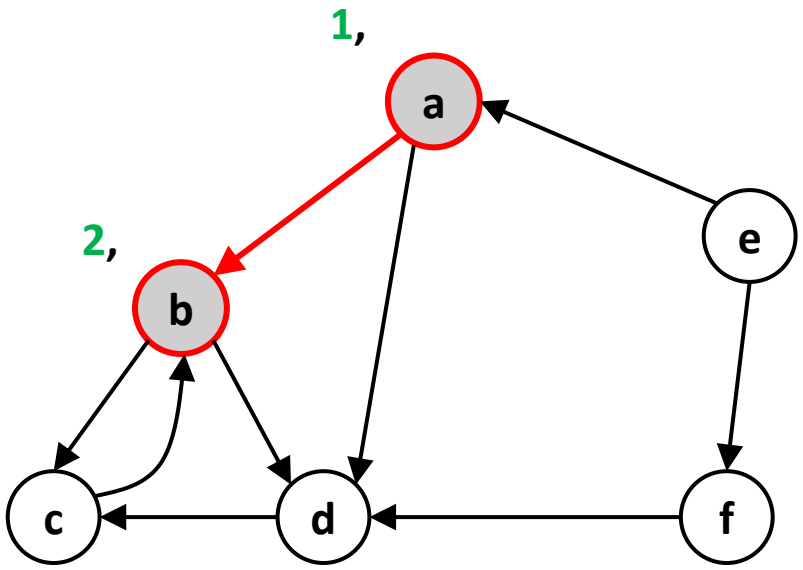


Colors of nodes:

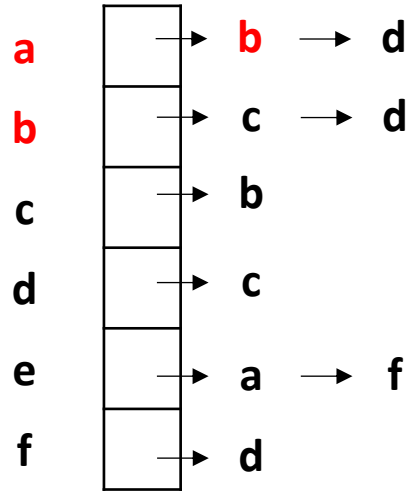
a	b	c	d	e	f
G	W	W	W	W	W

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

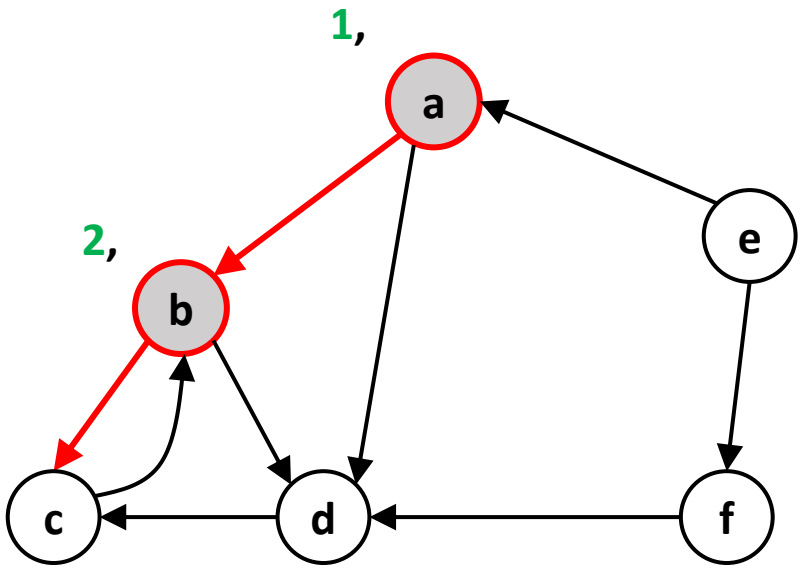


Colors of nodes:

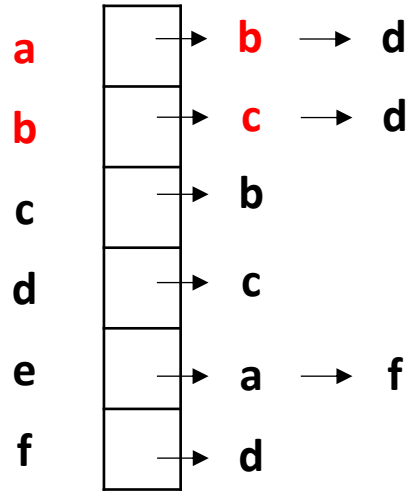
G	G	W	W	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

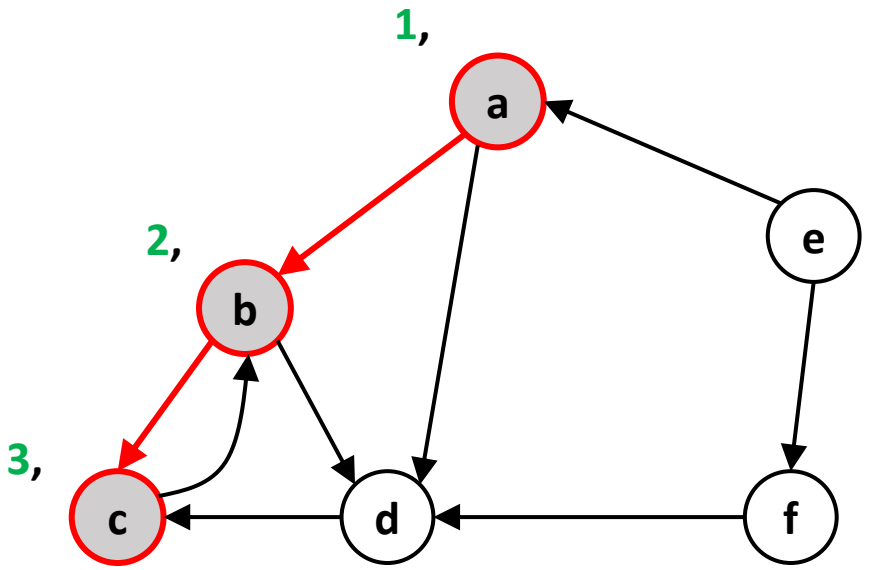


Colors of nodes:

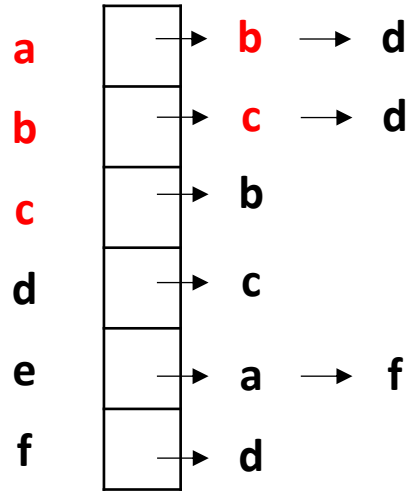
G	G	W	W	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

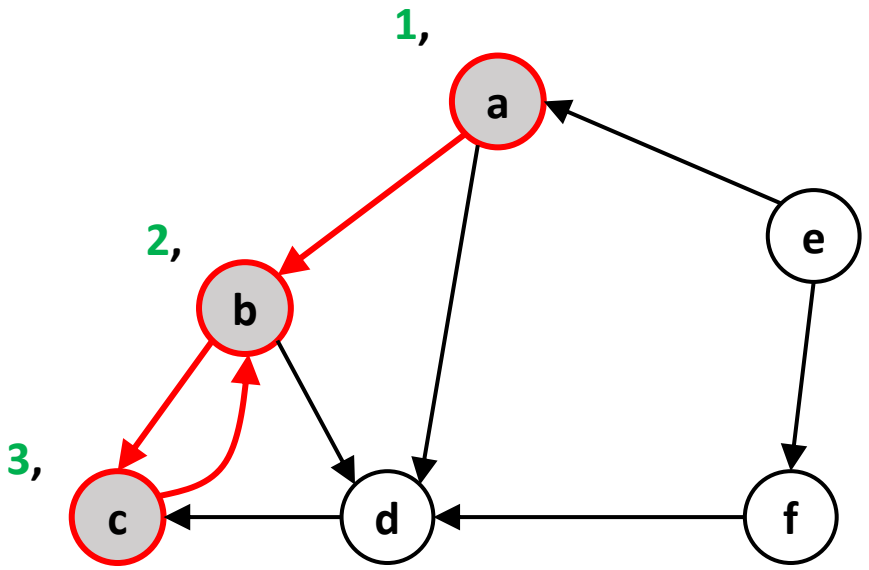


Colors of nodes:

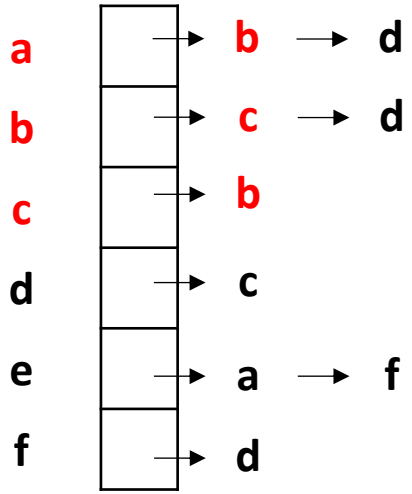
G	G	G	W	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

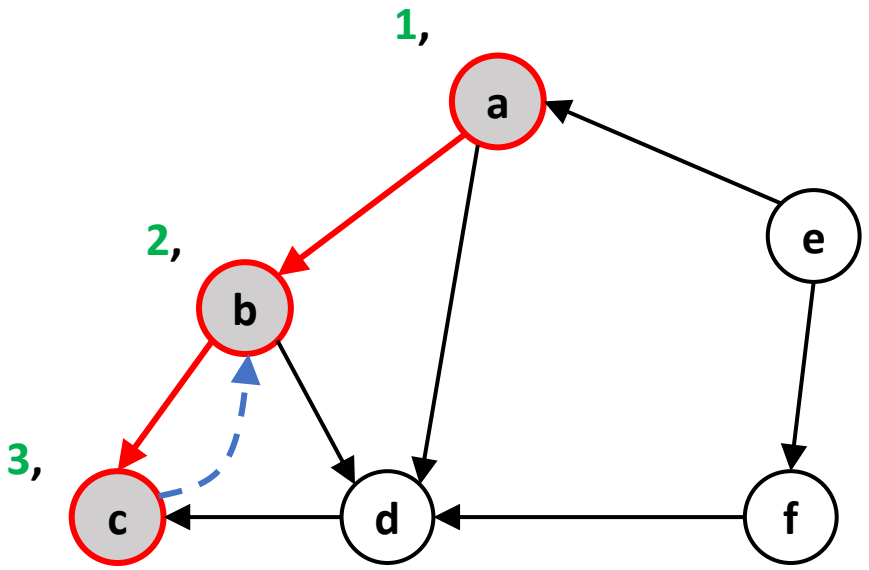


Colors of nodes:

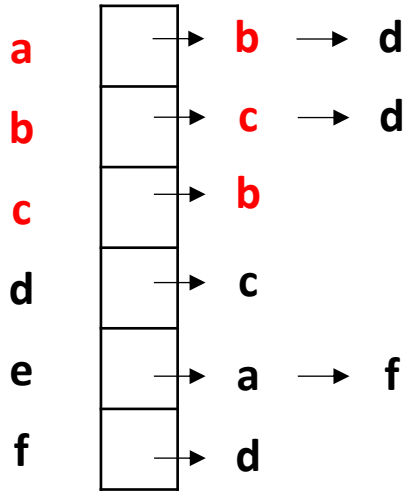
G	G	G	W	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

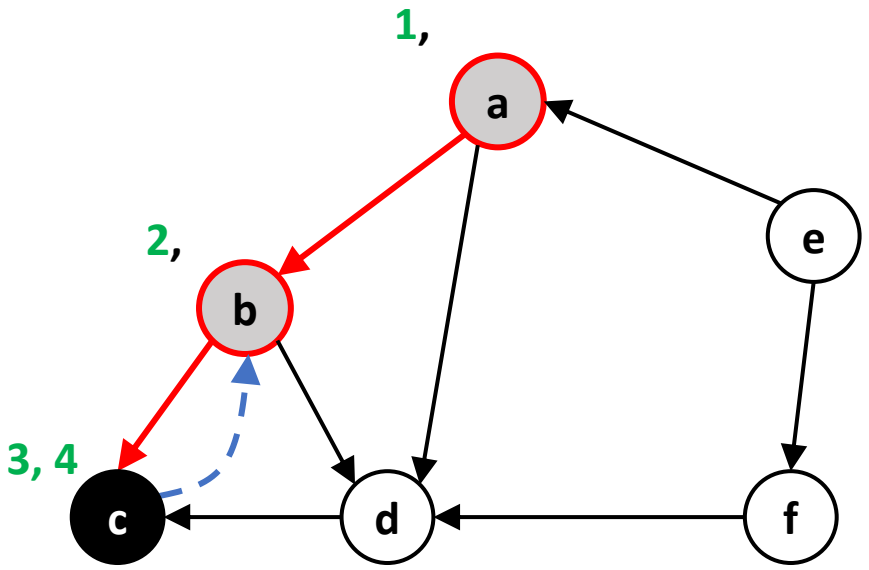


Colors of nodes:

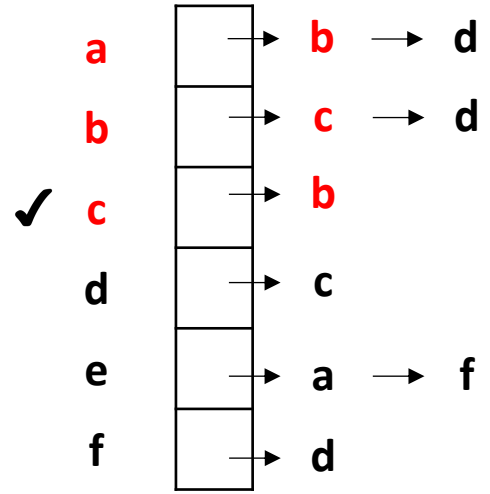
G	G	G	W	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

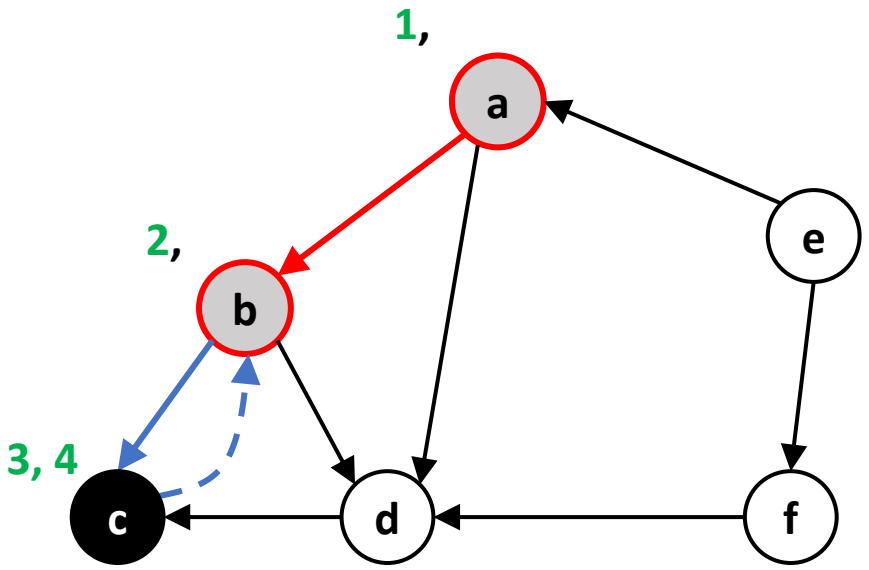


Colors of nodes:

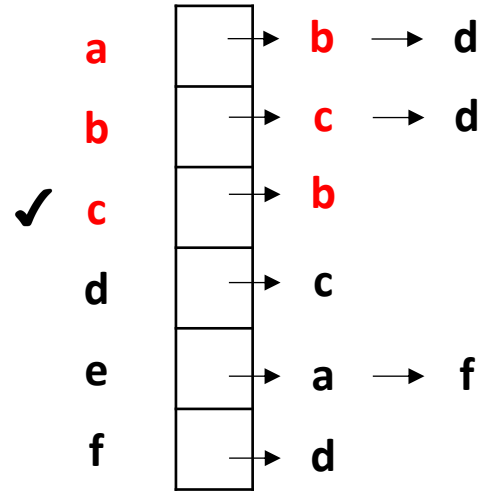
G	G	B	W	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

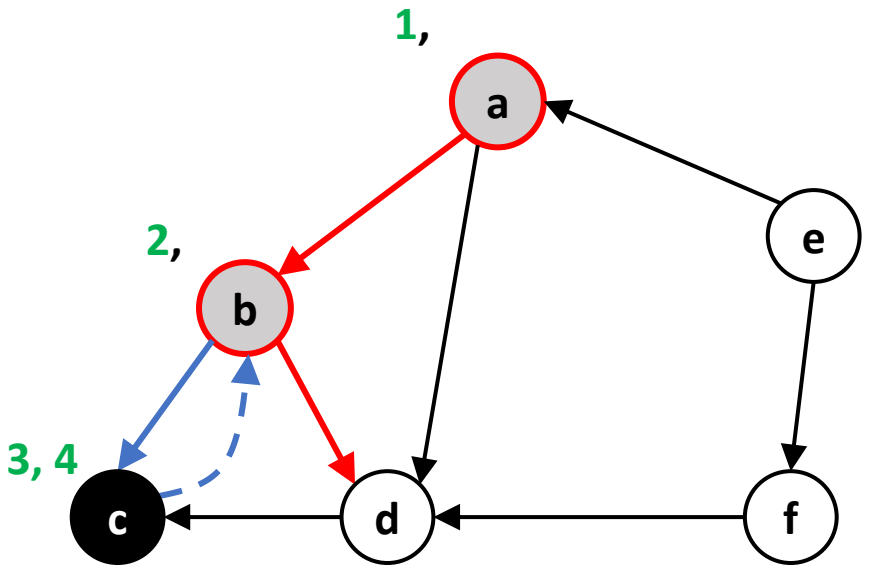


Colors of nodes:

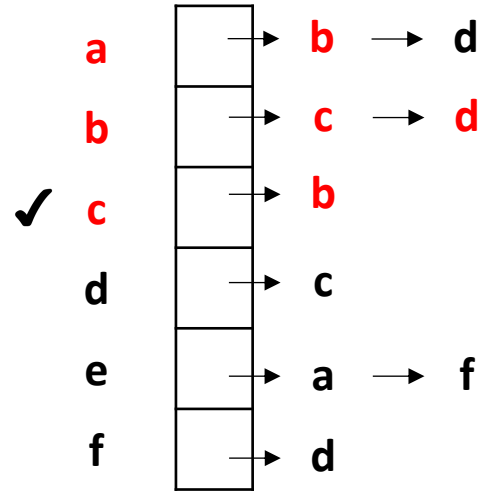
G	G	B	W	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

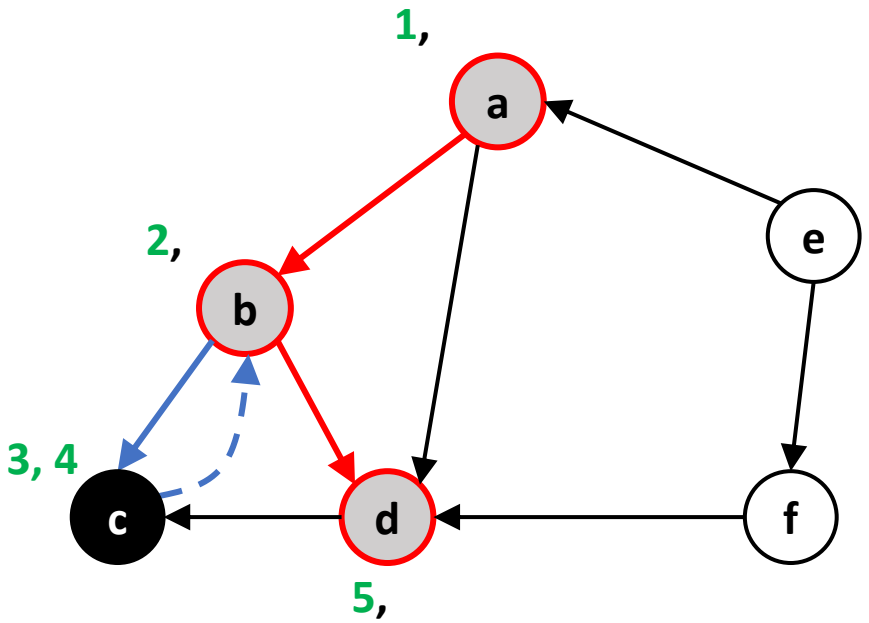


Colors of nodes:

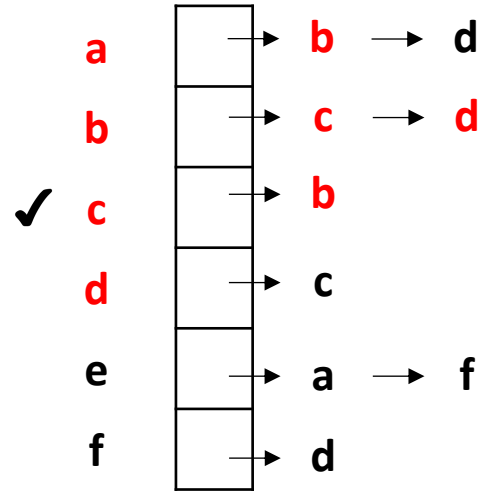
G	G	B	W	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

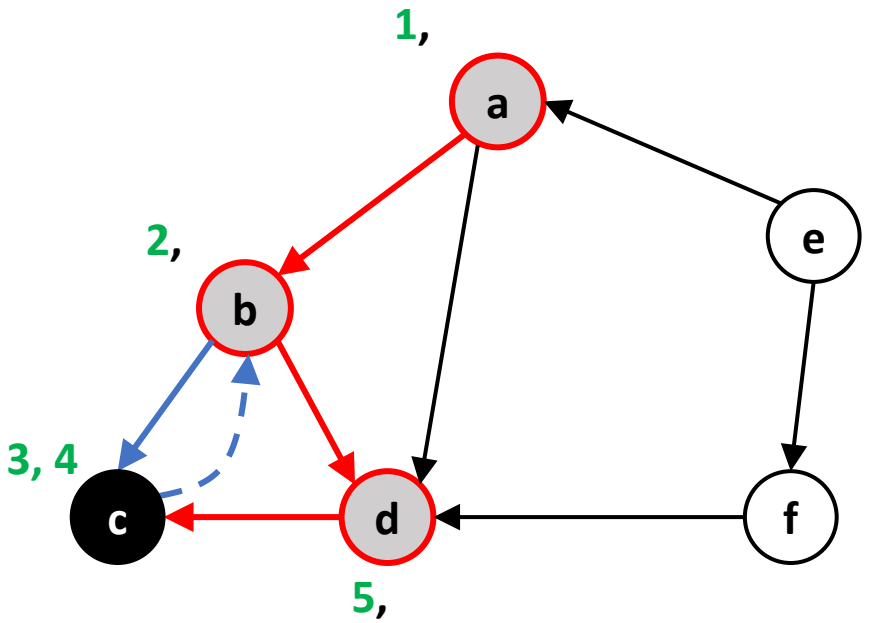


Colors of nodes:

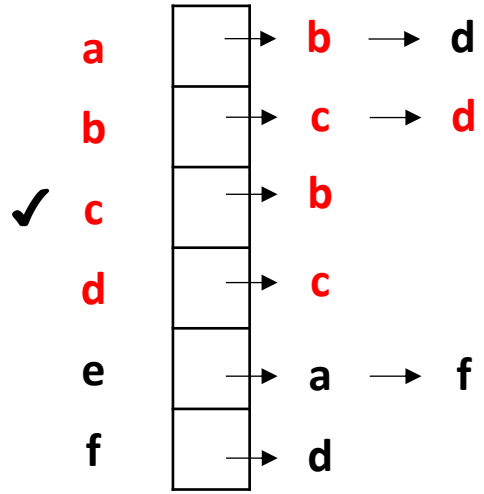
G	G	B	G	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

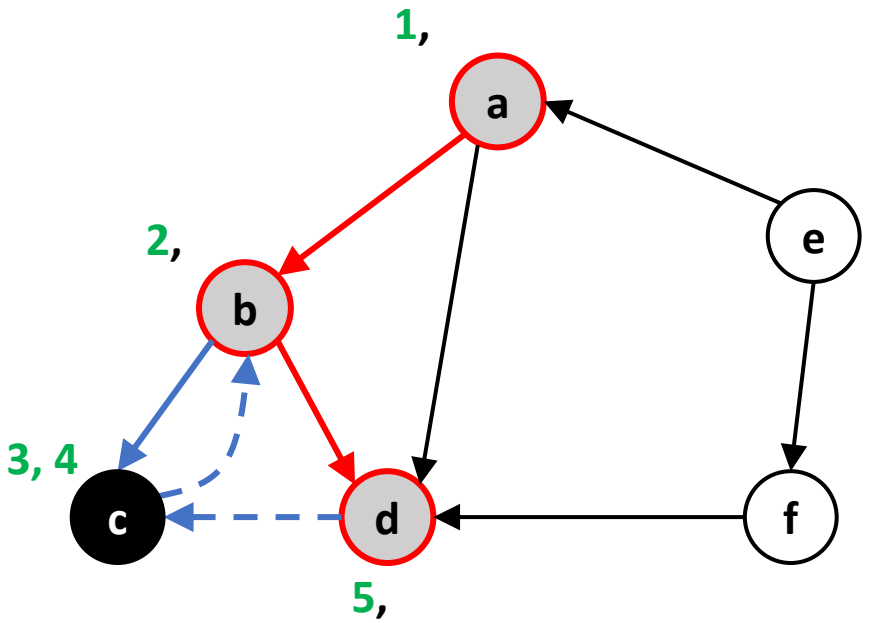


Colors of nodes:

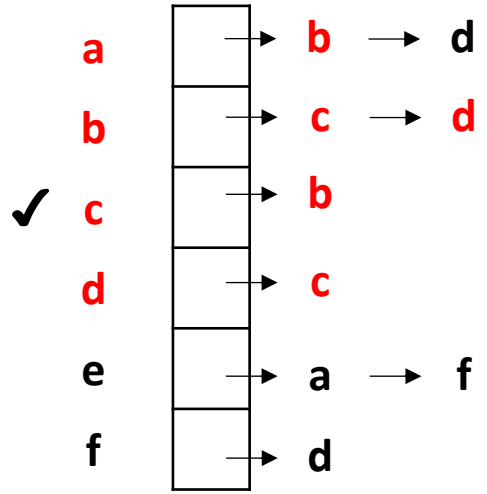
G	G	B	G	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

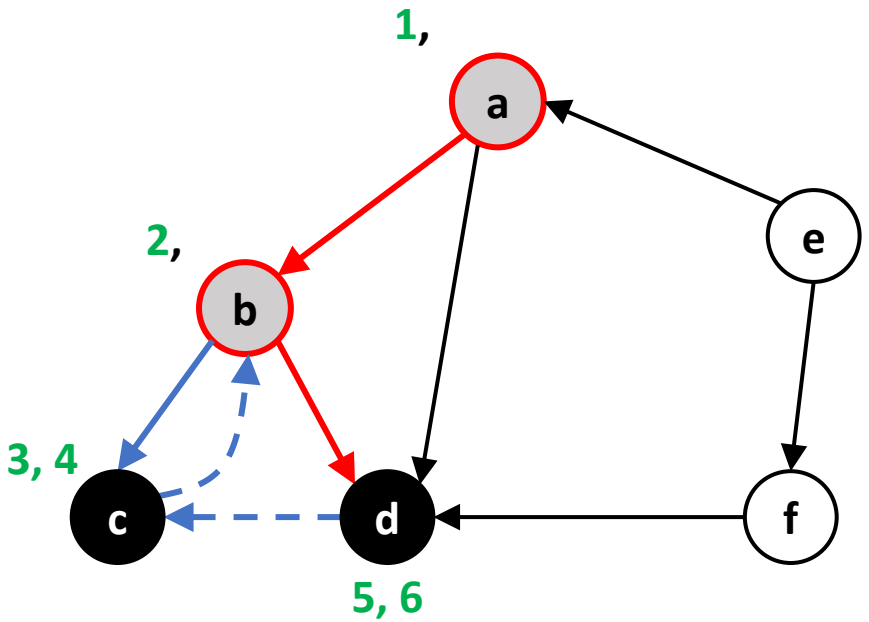


Colors of nodes:

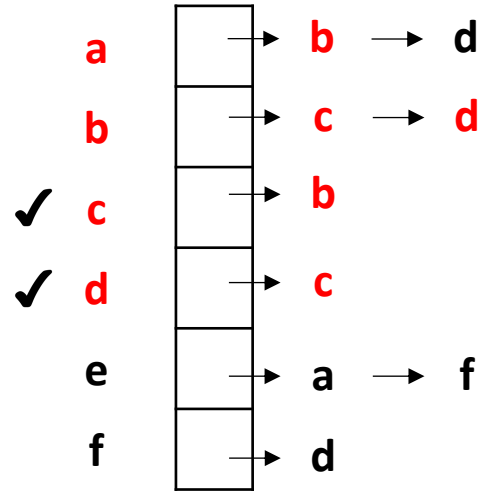
G	G	B	G	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

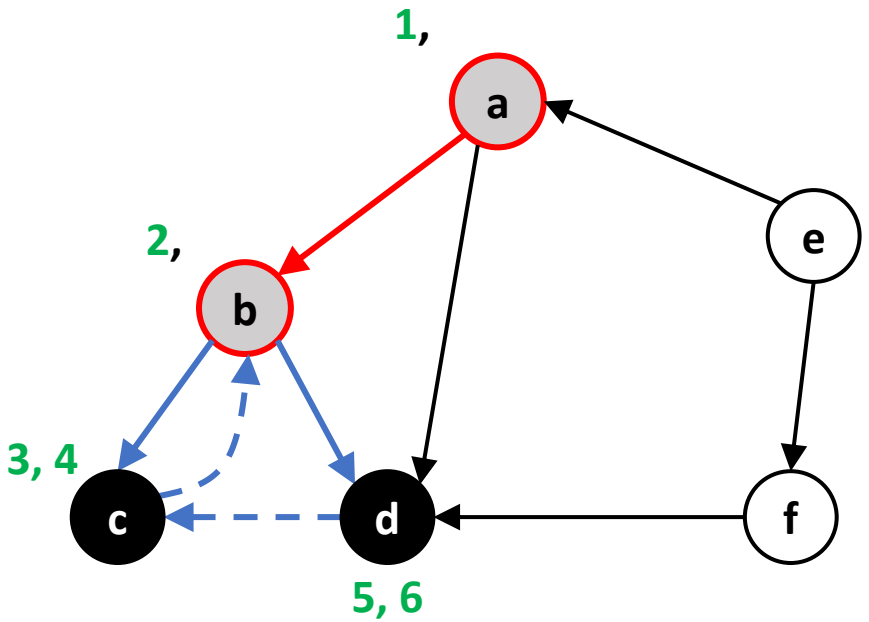


Colors of nodes:

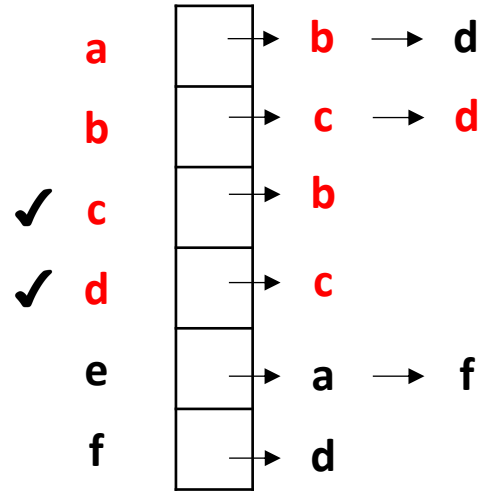
G	G	B	B	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

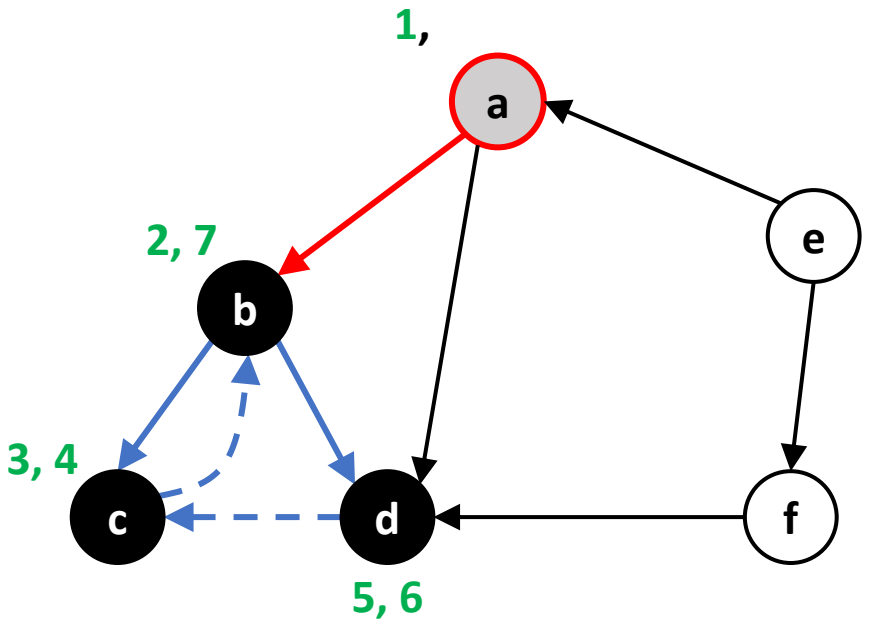


Colors of nodes:

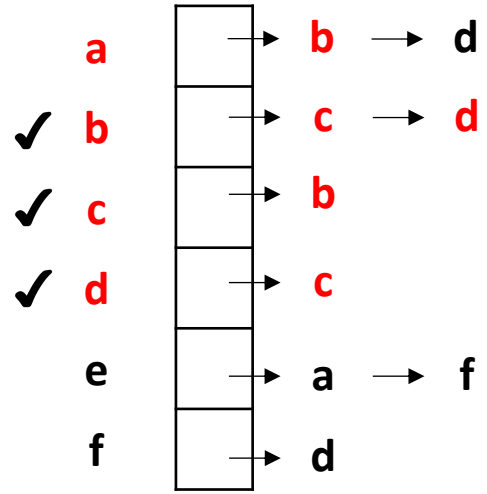
G	G	B	B	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

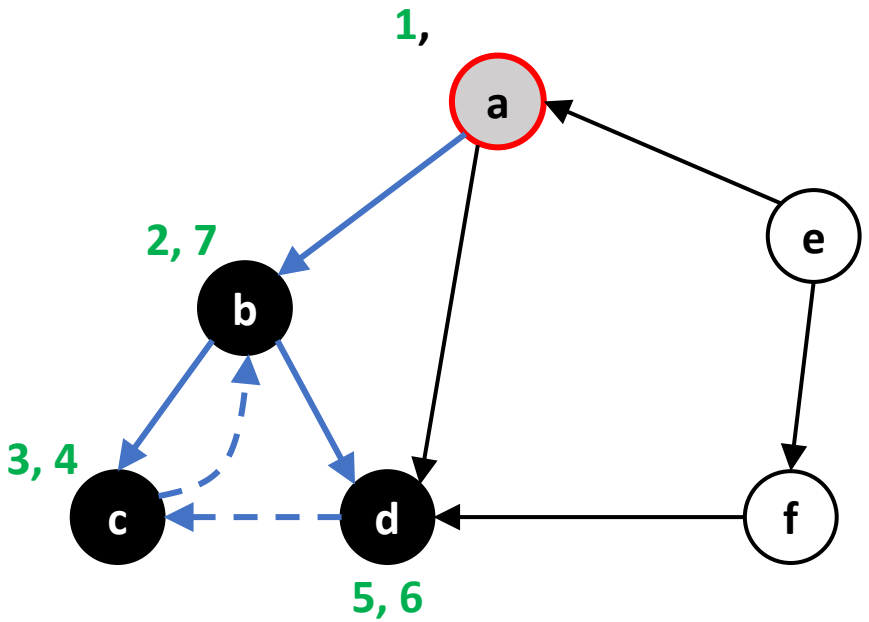


Colors of nodes:

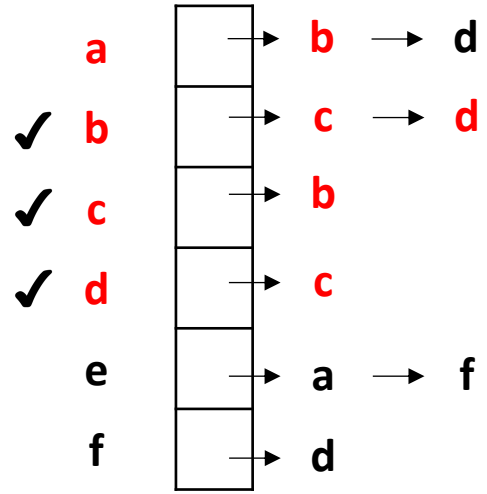
G	B	B	B	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

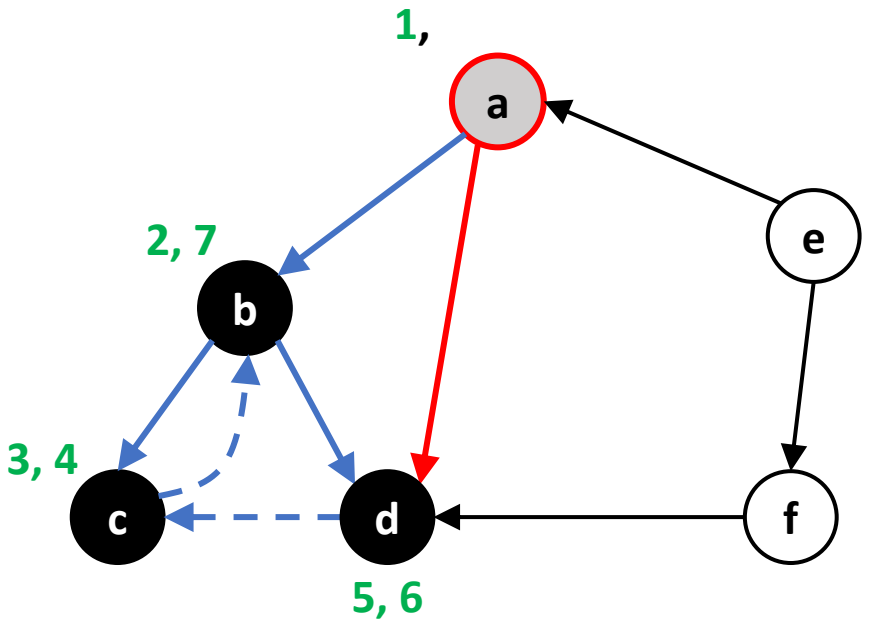


Colors of nodes:

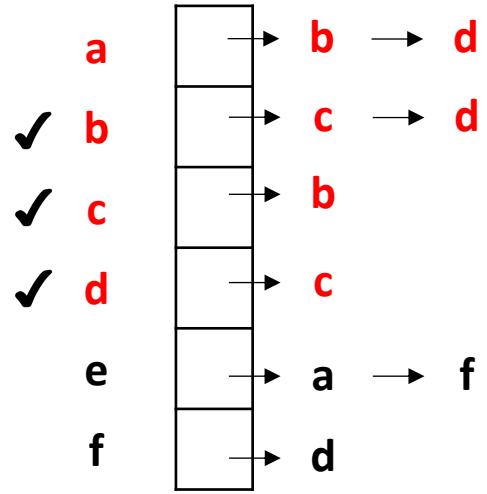
G	B	B	B	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

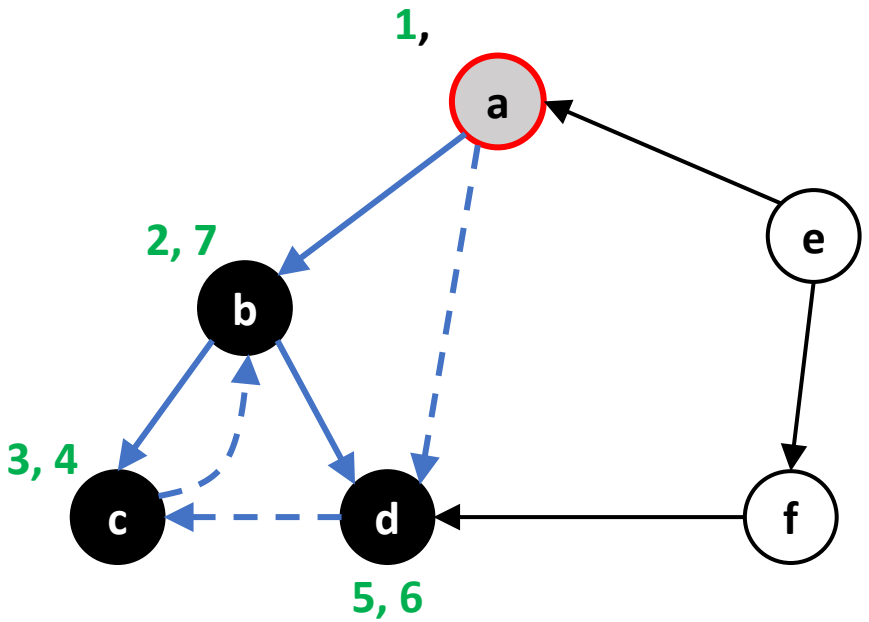


Colors of nodes:

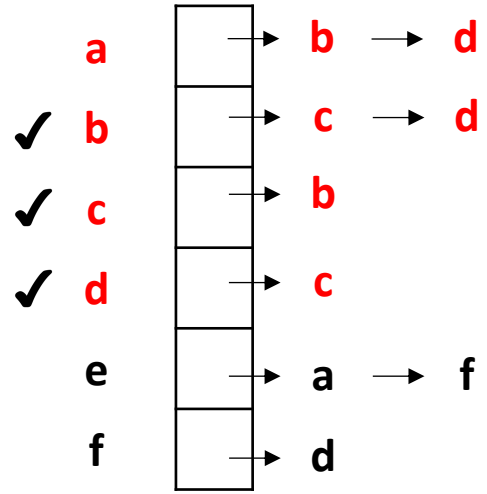
G	B	B	B	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

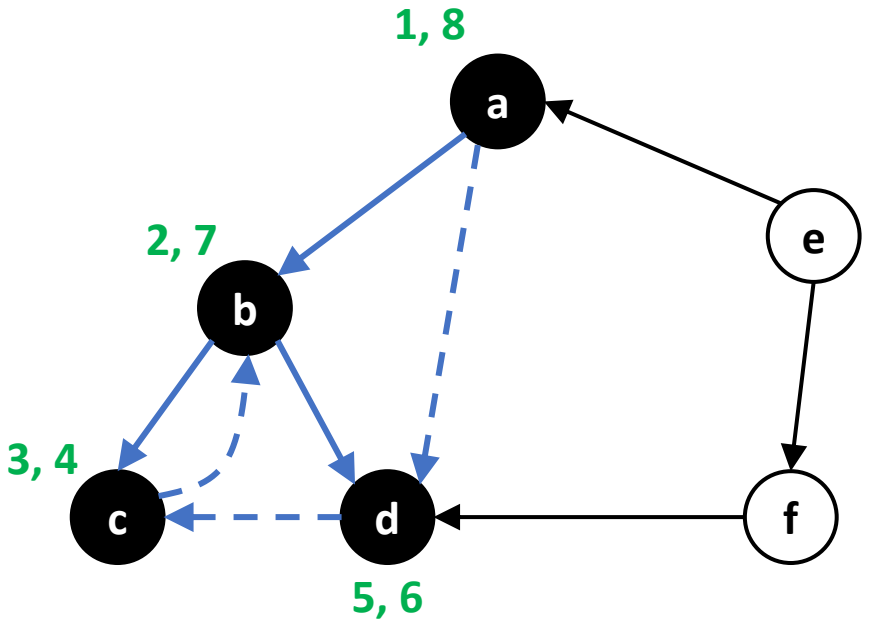


Colors of nodes:

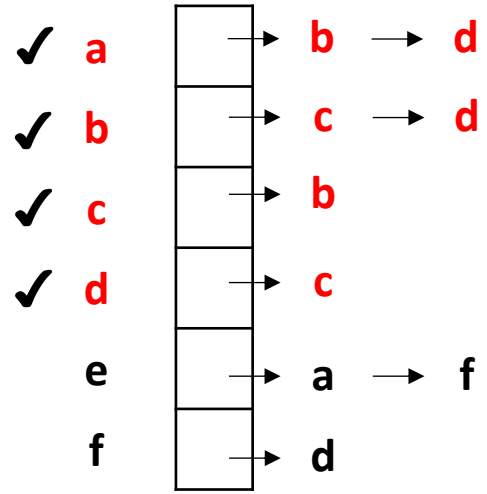
G	B	B	B	W	W
a	b	c	d	e	f

DFS-Explore(G, **a)**

DFS(G)



Adj List of G :

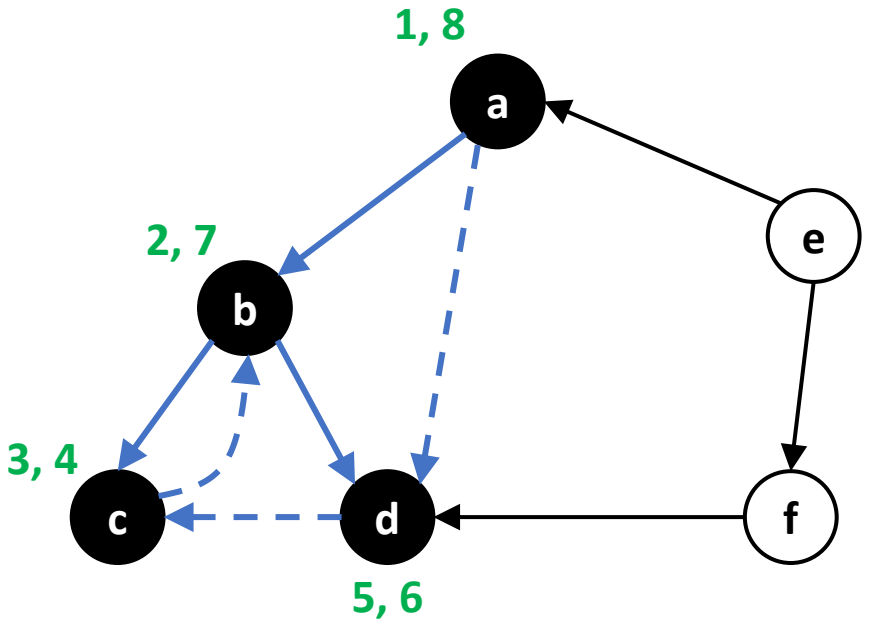


Colors of nodes:

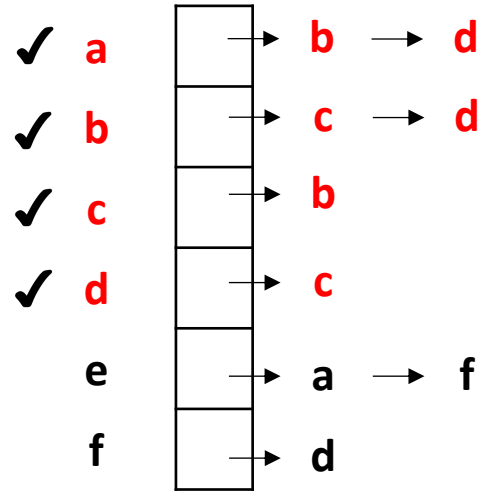
a	b	c	d	e	f
B	B	B	B	W	W

DFS-Explore(G, **a)**

DFS(G)



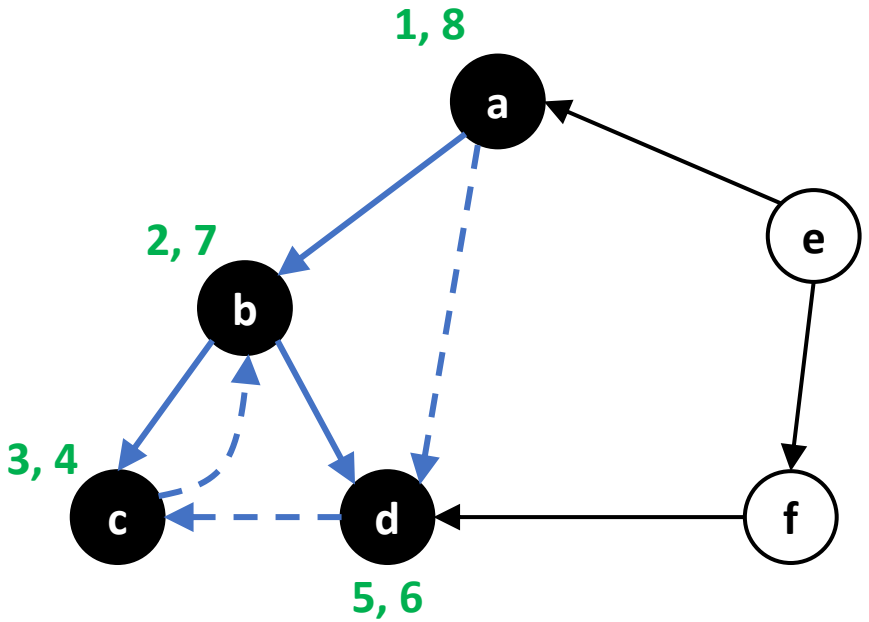
Adj List of G :



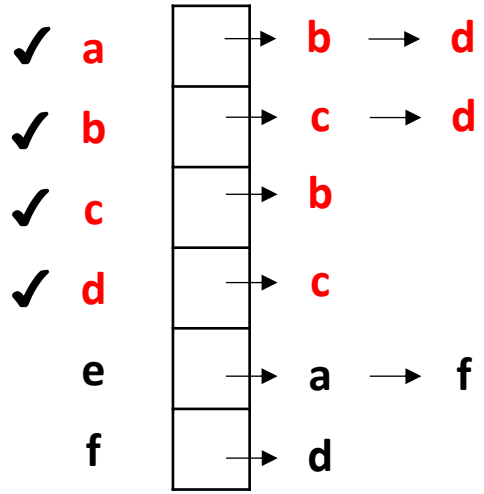
Colors of nodes:

a	b	c	d	e	f
B	B	B	B	W	W

DFS(G)



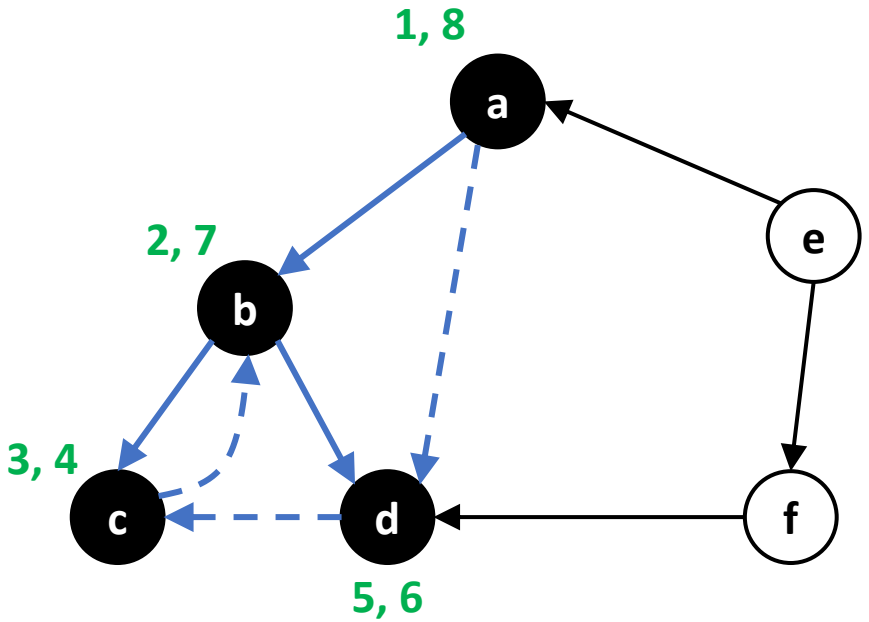
Adj List of G :



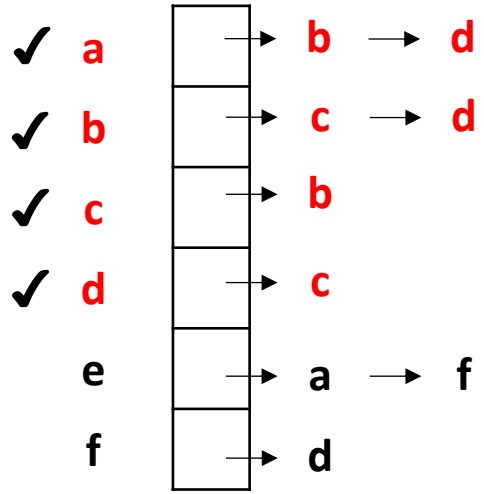
Colors of nodes:

a	b	c	d	e	f
B	B	B	B	W	W

DFS(G)



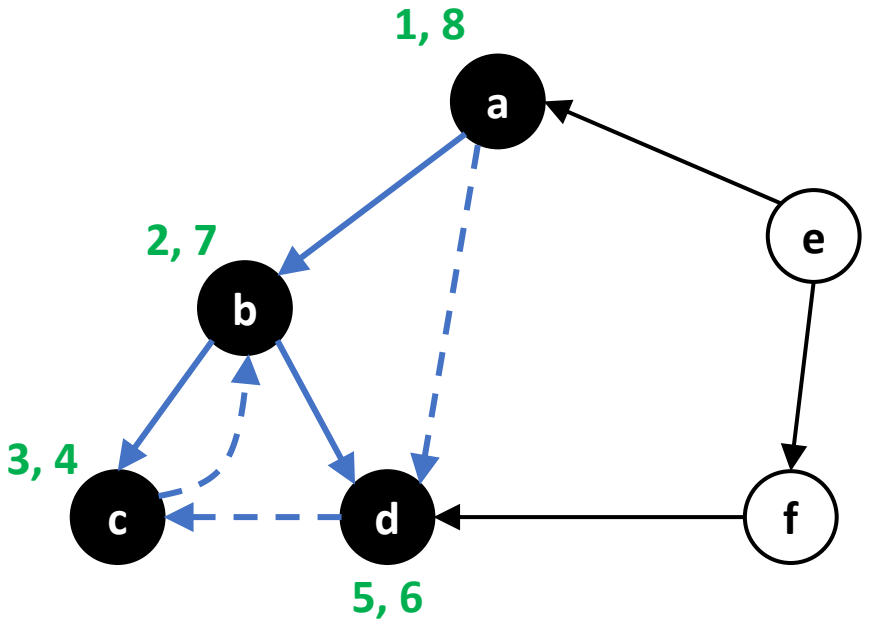
Adj List of G :



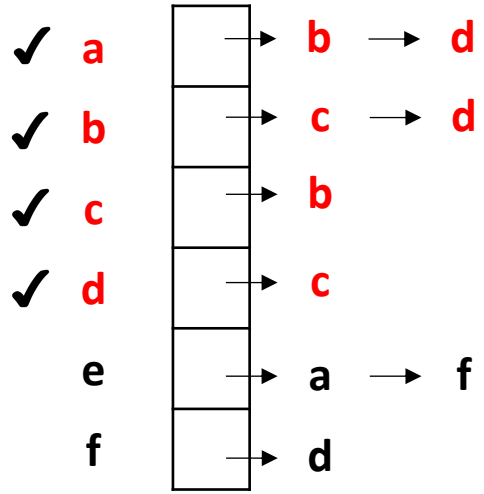
Colors of nodes:

B	B	B	B	W	W
a	b	c	d	e	f

DFS(G)



Adj List of G :

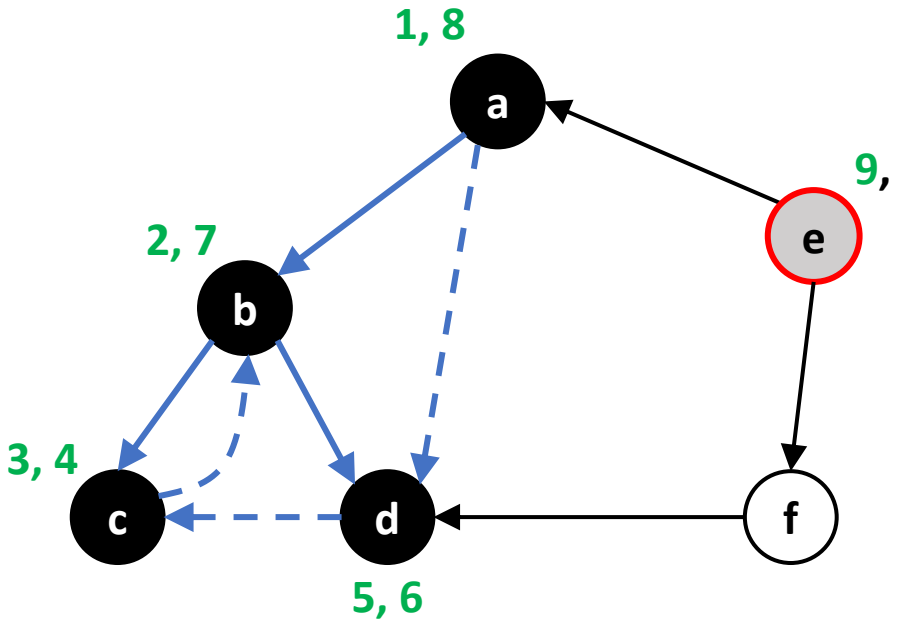


Colors of nodes:

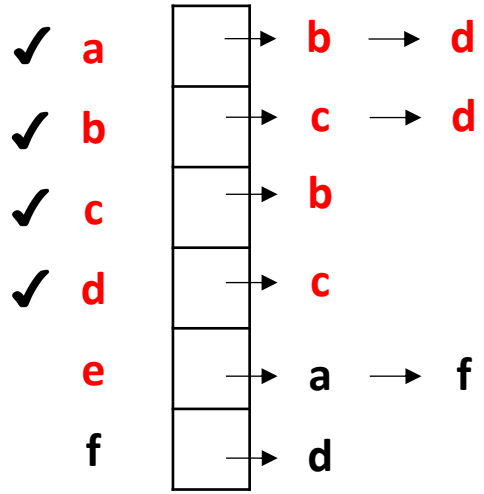
B	B	B	B	W	W
a	b	c	d	e	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

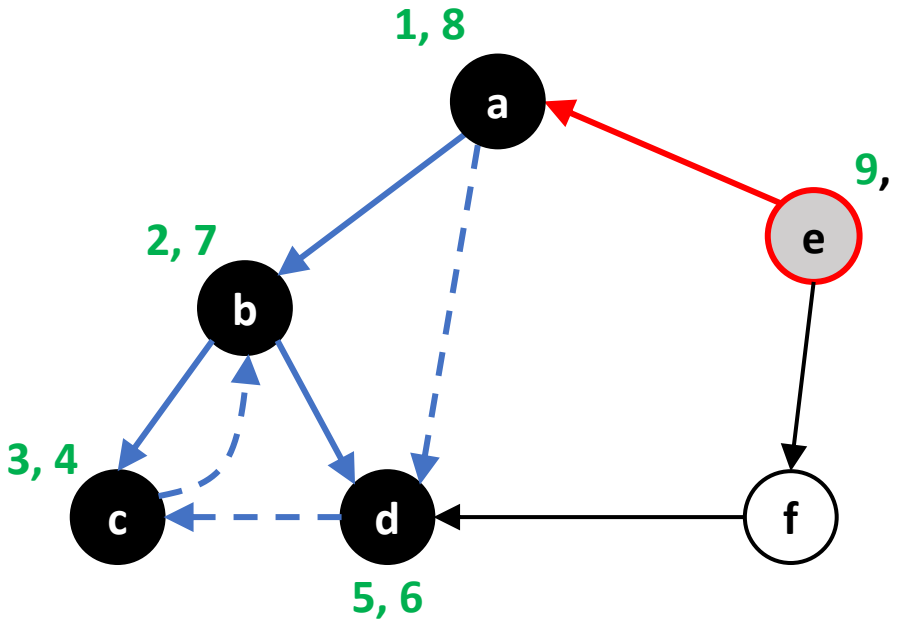


Colors of nodes:

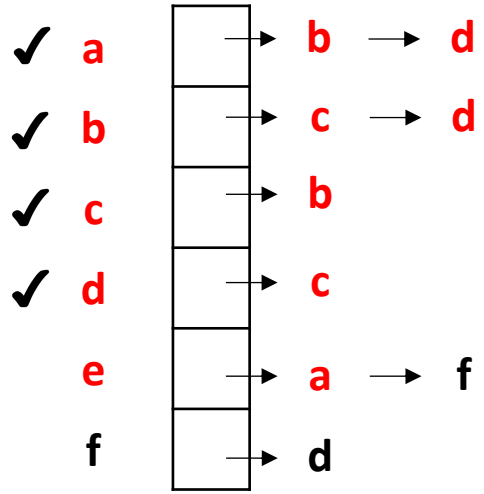
B	B	B	B	G	W
a	b	c	d	e	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

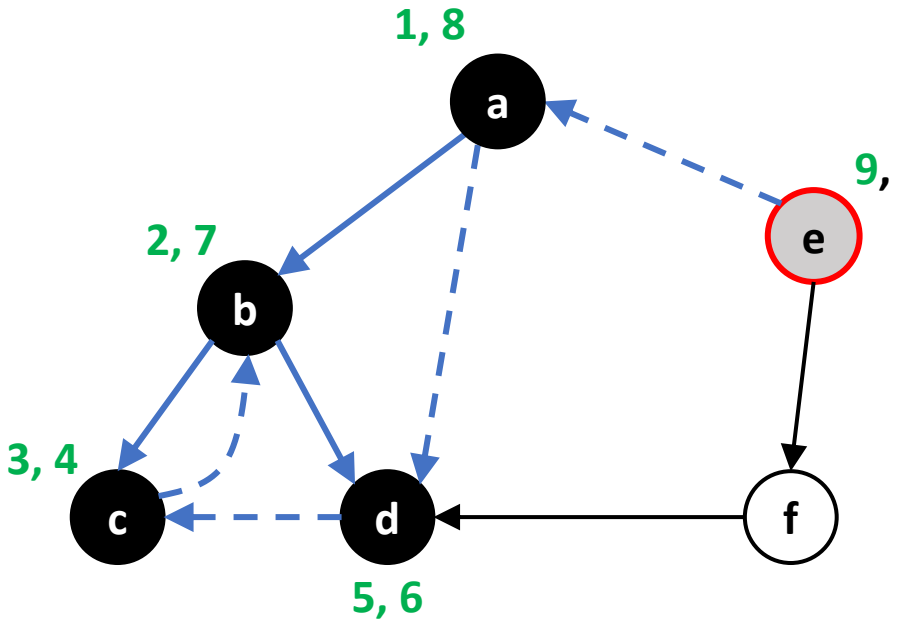


Colors of nodes:

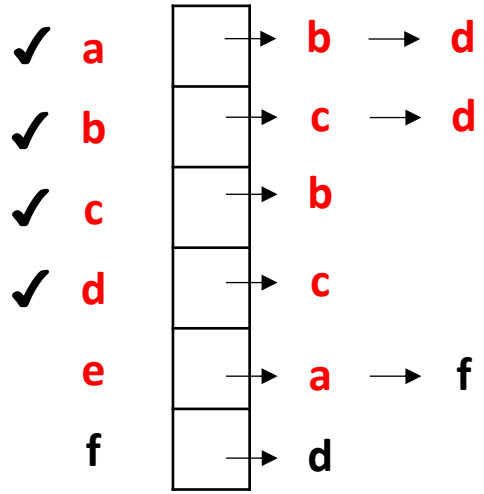
B	B	B	B	G	W
a	b	c	d	e	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

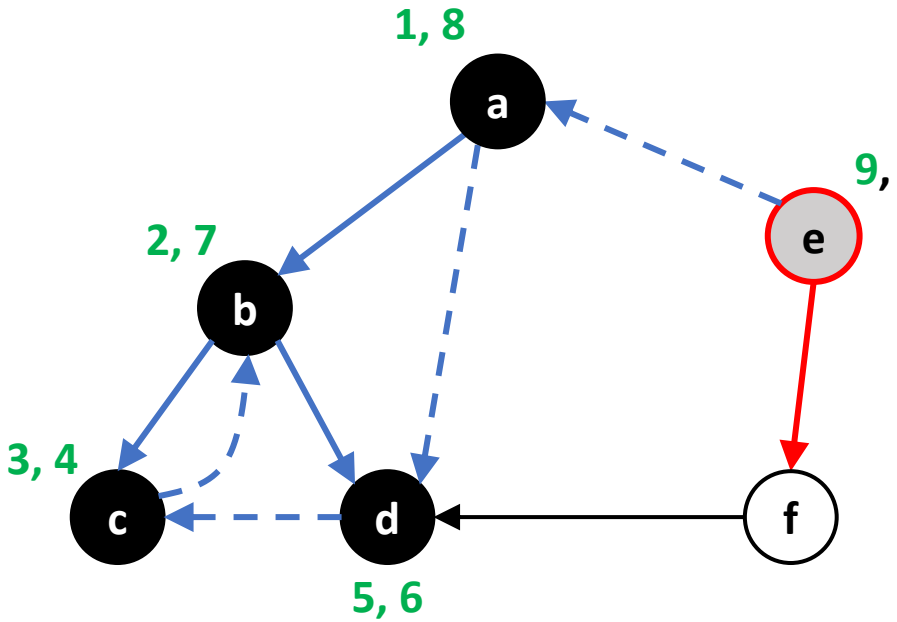


Colors of nodes:

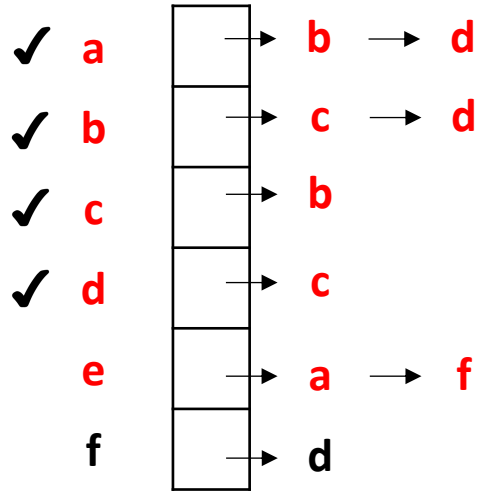
a	b	c	d	e	f
B	B	B	B	G	W

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

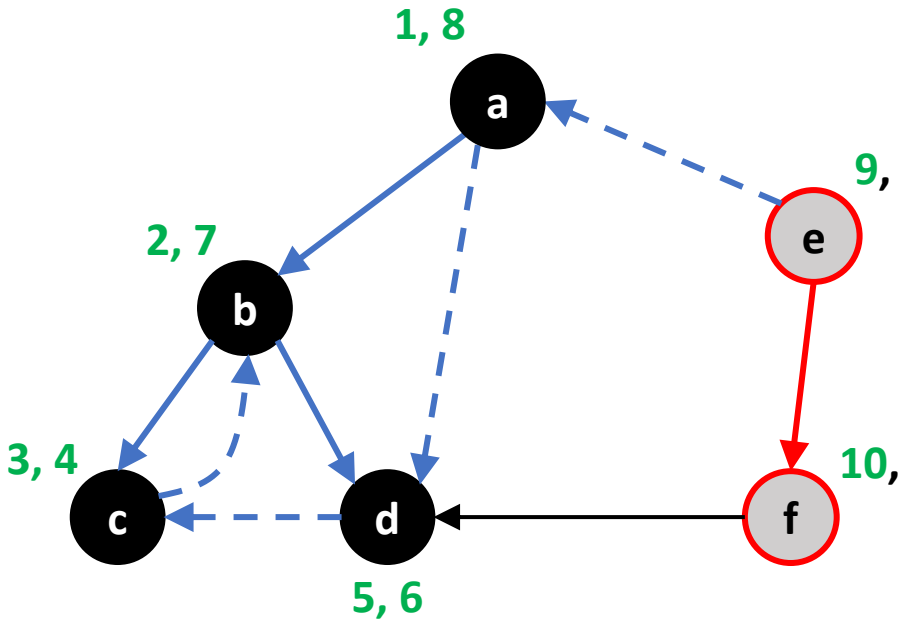


Colors of nodes:

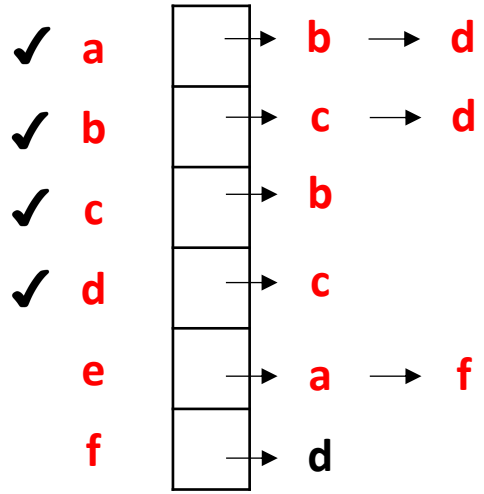
a	b	c	d	e	f
B	B	B	B	G	W

DFS-Explore(G, **e)**

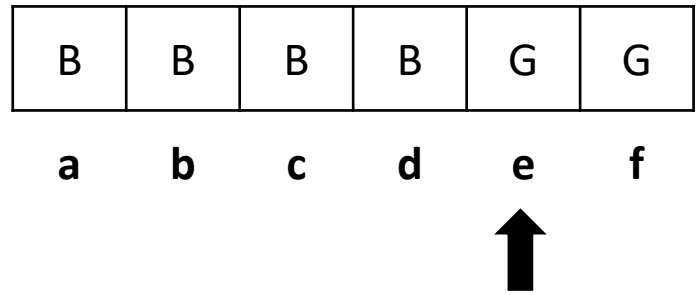
DFS(G)



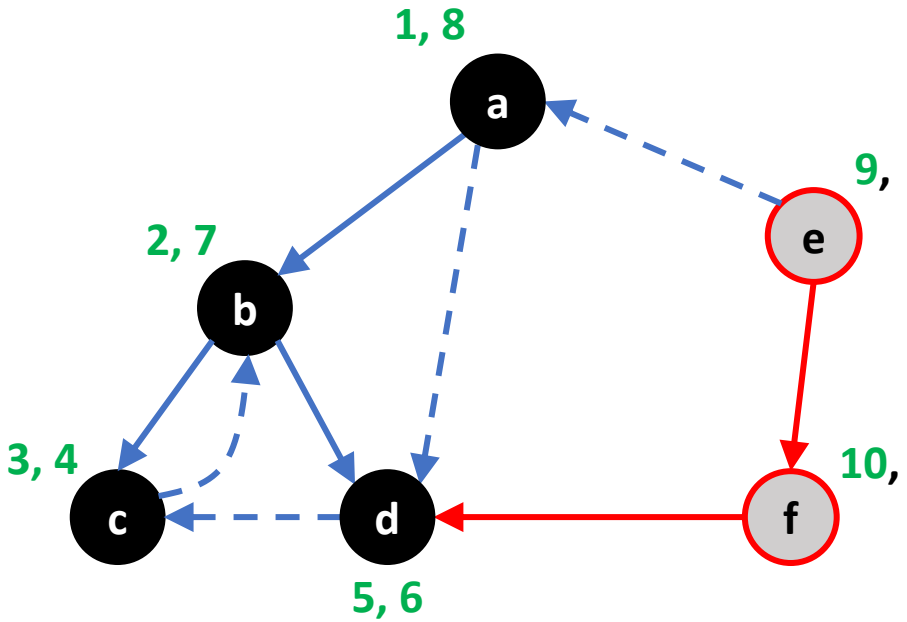
Adj List of G :



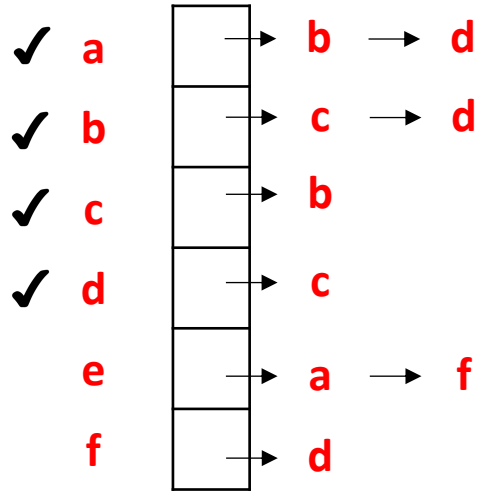
Colors of nodes:



DFS(G)



Adj List of G :

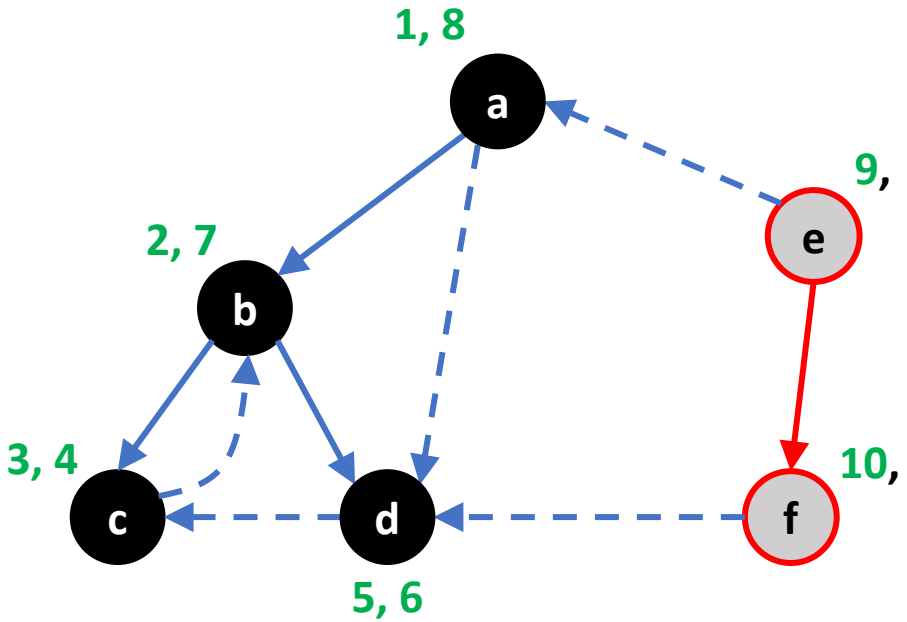


Colors of nodes:

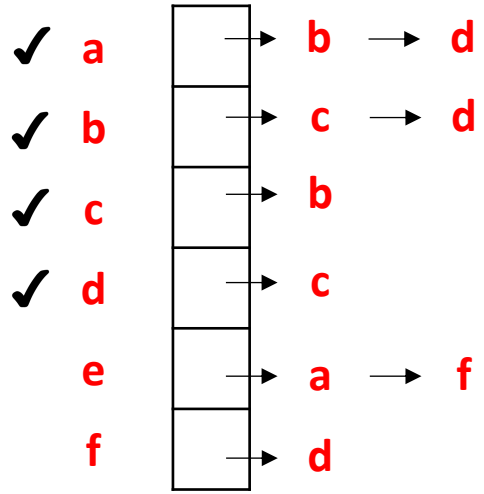
B	B	B	B	G	G
a	b	c	d	e	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

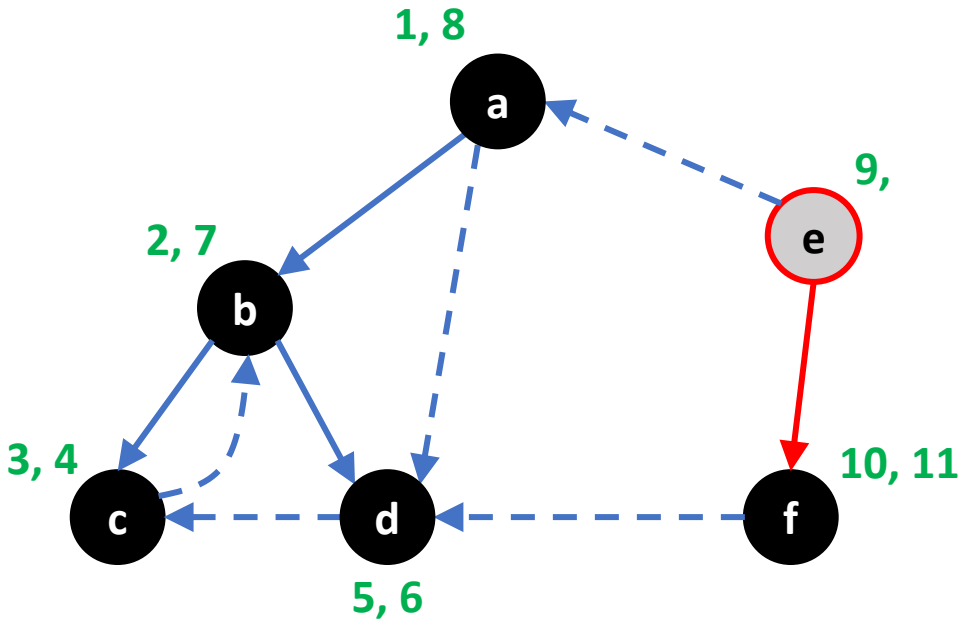


Colors of nodes:

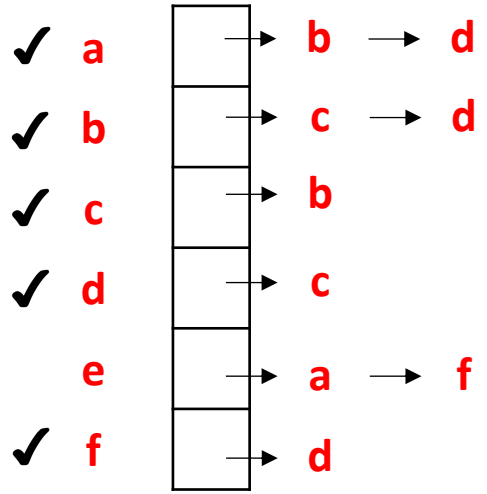
B	B	B	B	G	G
a	b	c	d	e	f

DFS-Explore(G, **e)**

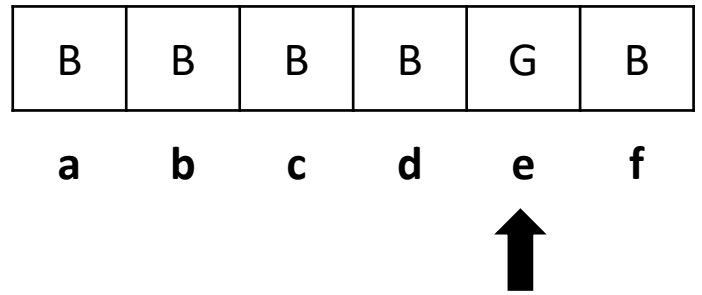
DFS(G)



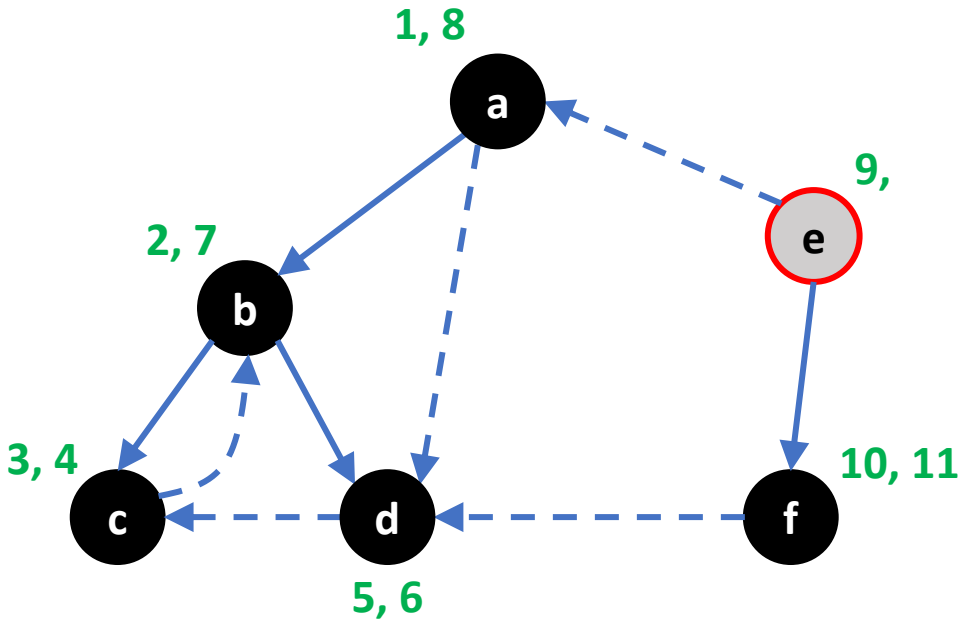
Adj List of G :



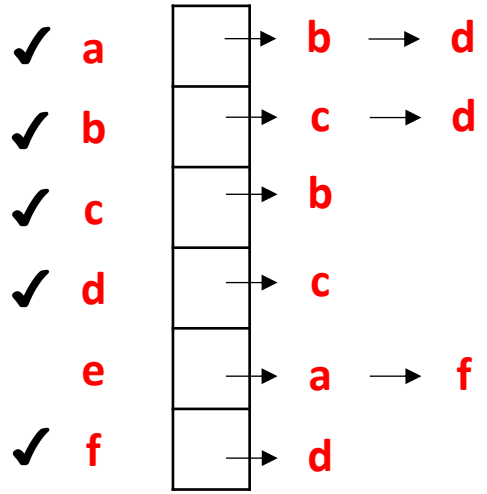
Colors of nodes:



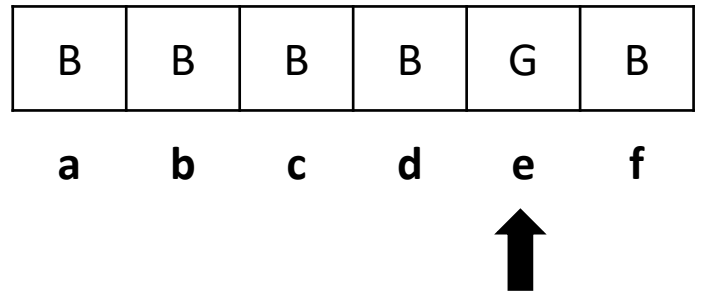
DFS(G)



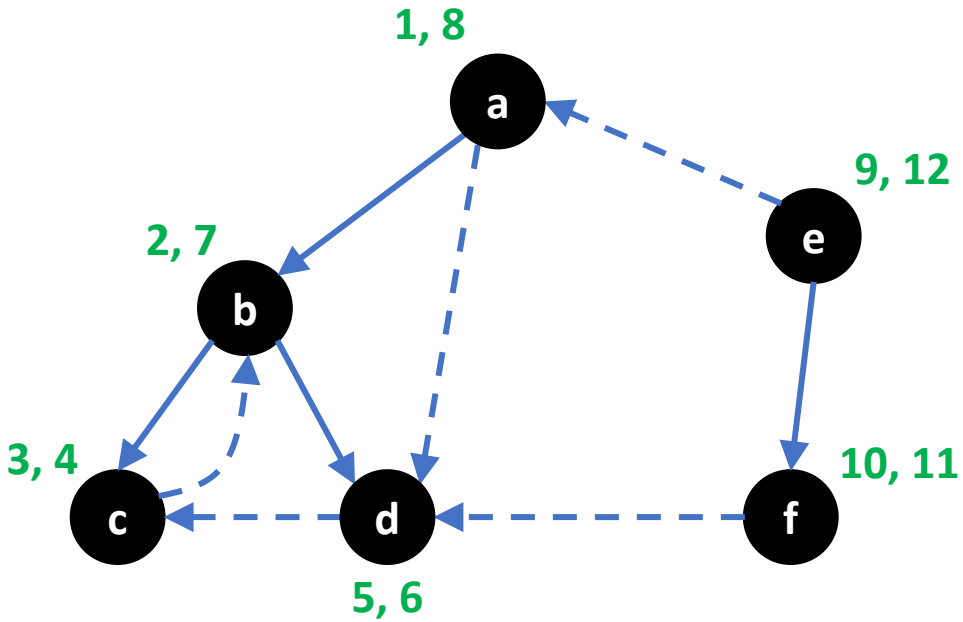
Adj List of G :



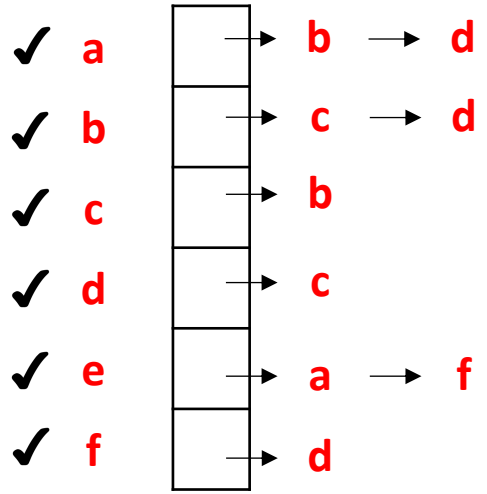
Colors of nodes:



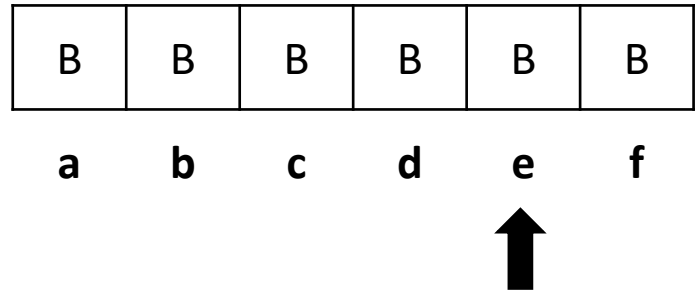
DFS(G)



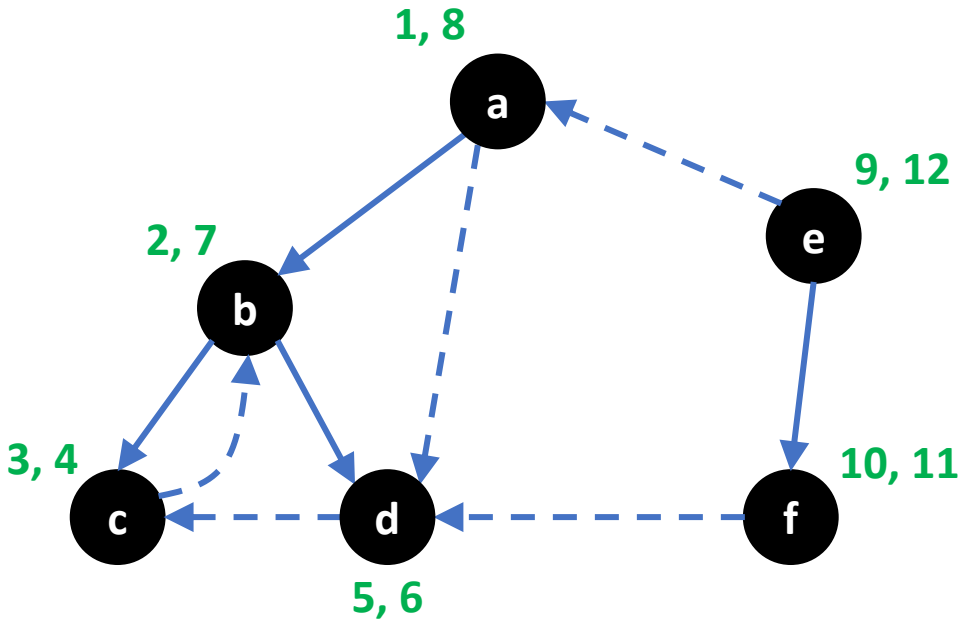
Adj List of G :



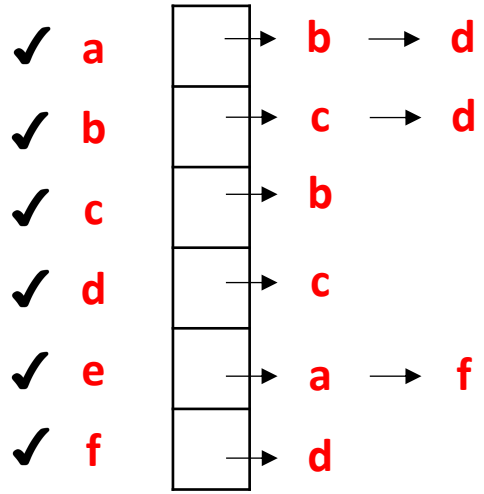
Colors of nodes:



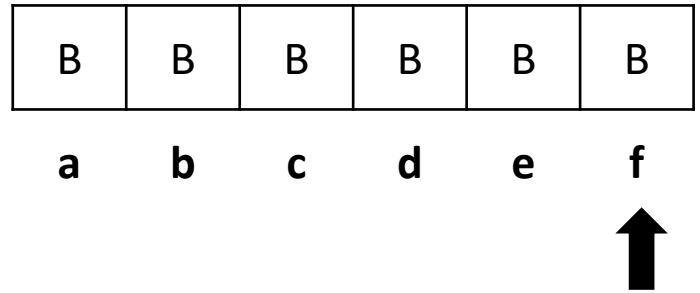
DFS(G)



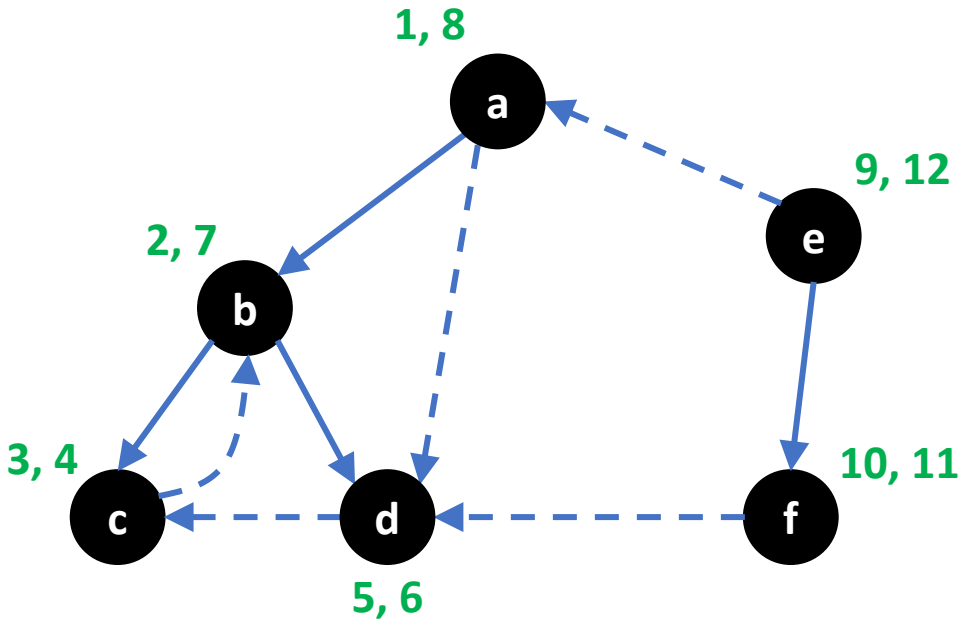
Adj List of G :



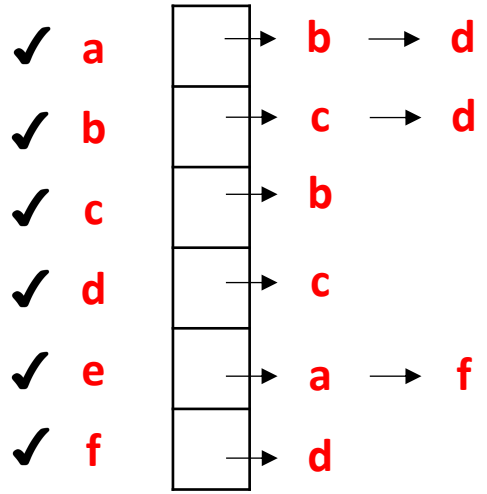
Colors of nodes:



DFS(G)



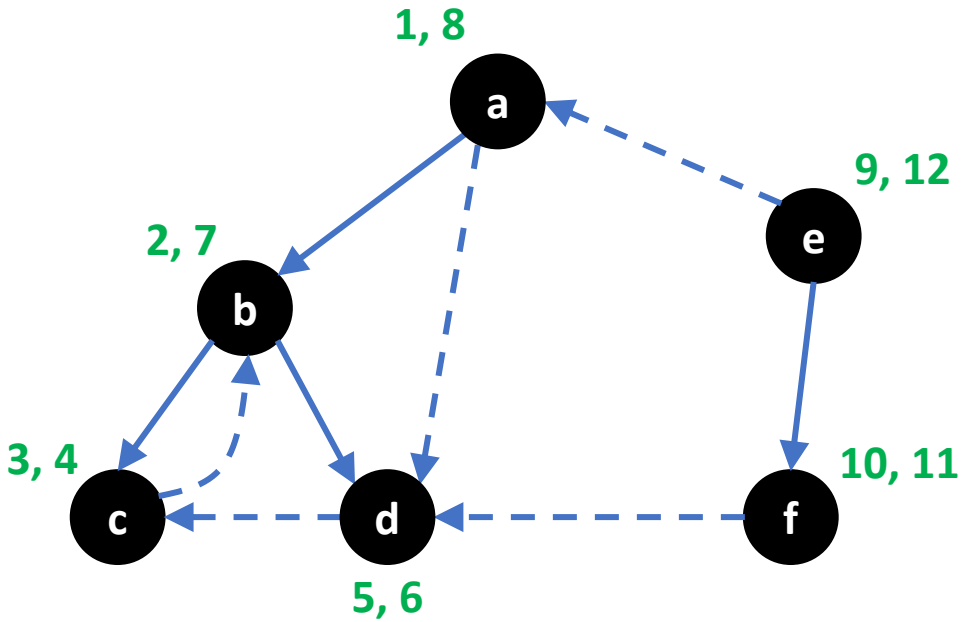
Adj List of G :



Colors of nodes:

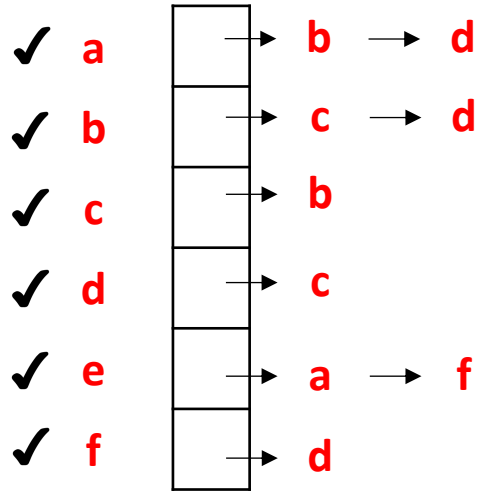
B	B	B	B	B	B
a	b	c	d	e	f

DFS(G)



Worst-case time complexity:

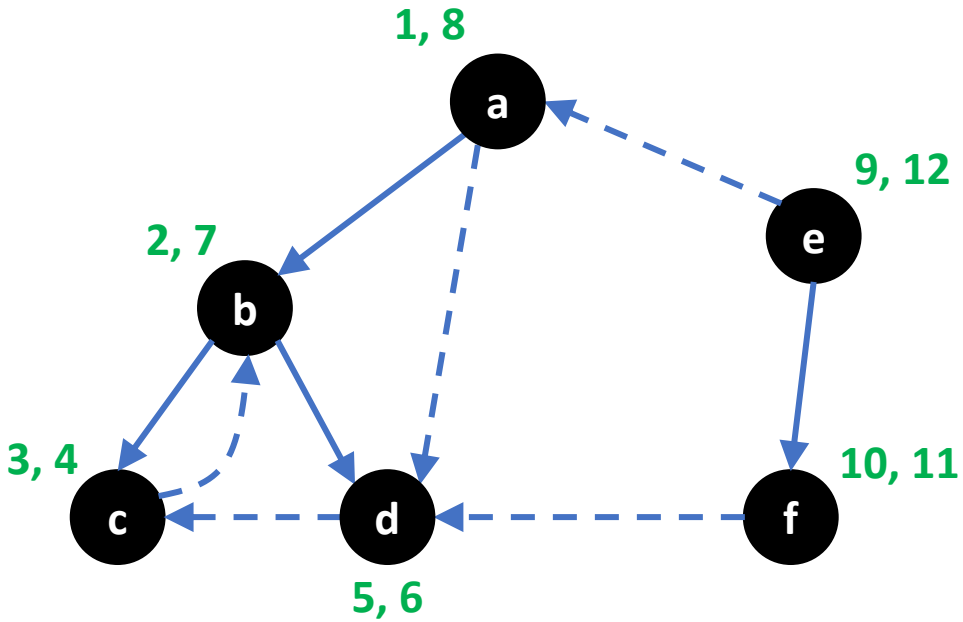
Adj List of G :



Colors of nodes:

B	B	B	B	B	B
a	**b**	**c**	**d**	**e**	**f**

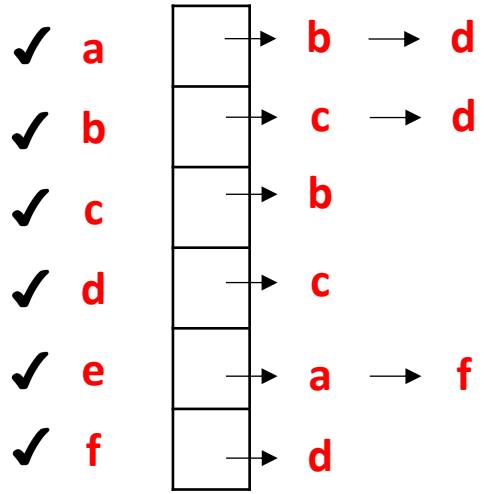
DFS(G)



Worst-case time complexity: O(|V| + |E|)

O(n + m)

Adj List of G :

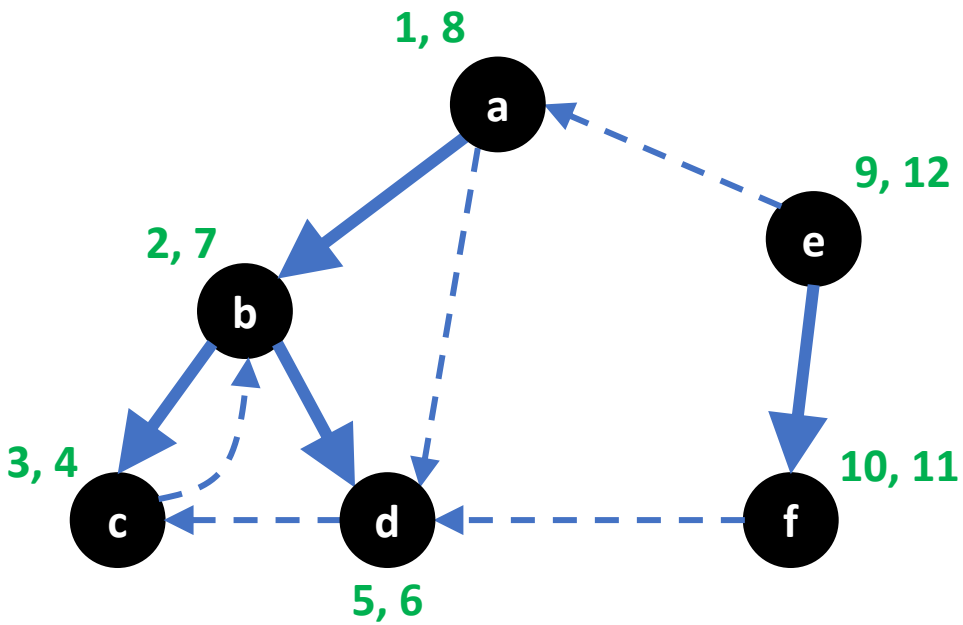


Colors of nodes:

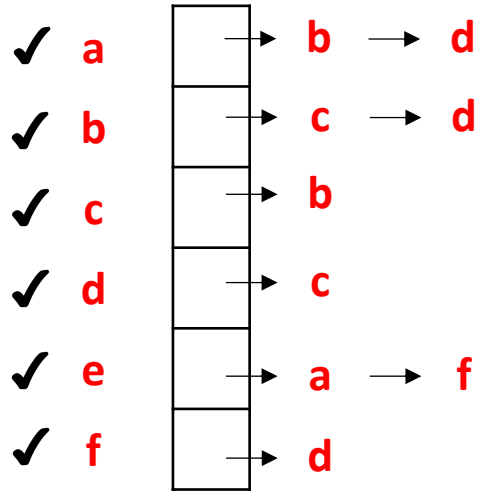
a	b	c	d	e	f
B	B	B	B	B	B

DFS(G)

DFS Forest : **solid** edges



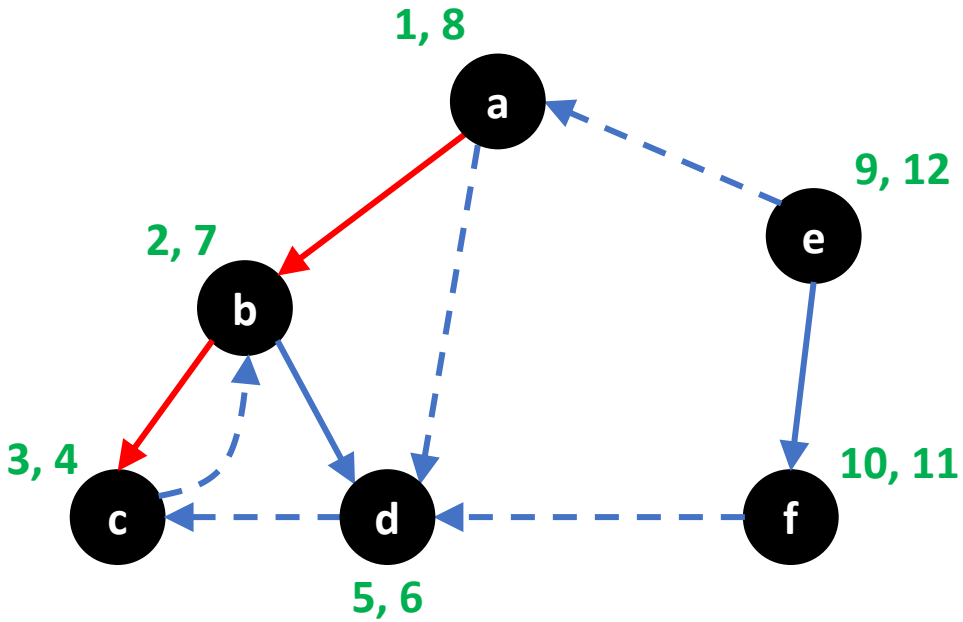
Adj List of G :



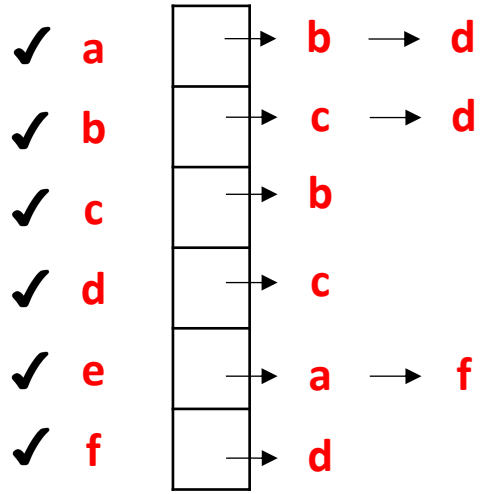
Colors of nodes:

B	B	B	B	B	B
a	**b**	**c**	**d**	**e**	**f**

DFS(G)



Adj List of G :

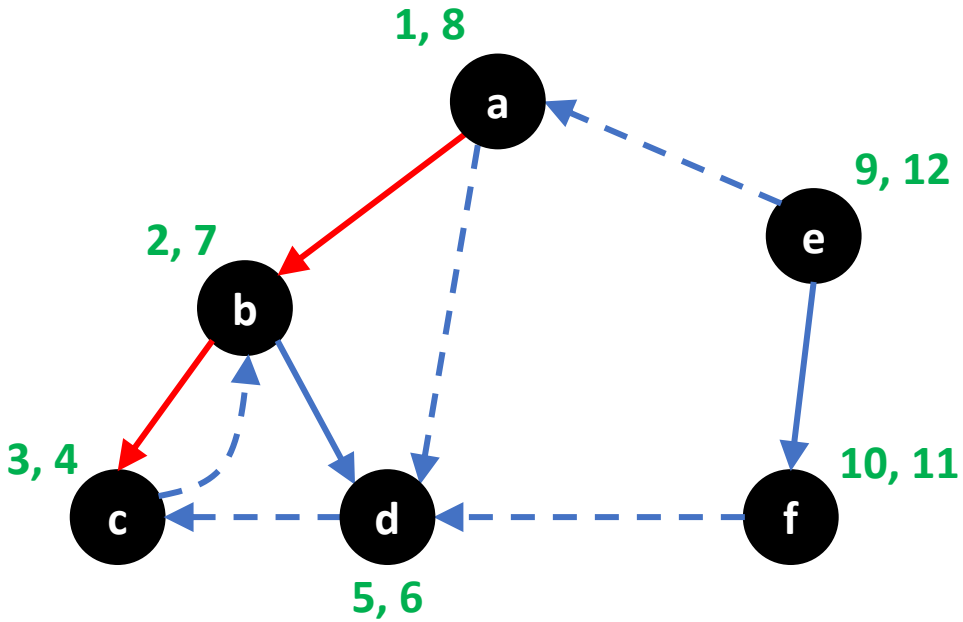


Colors of nodes:

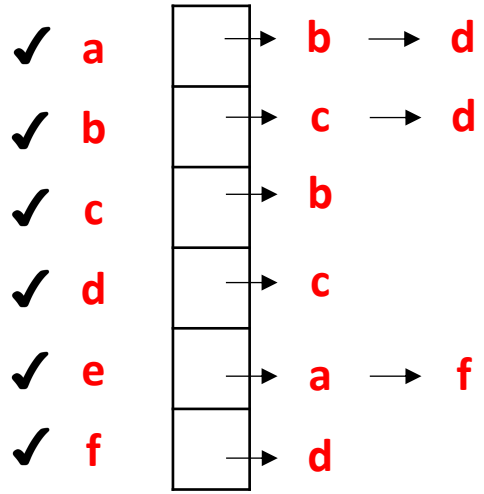
B	B	B	B	B	B
a	**b**	**c**	**d**	**e**	**f**

a is **ancestor** of c

DFS(G)



Adj List of G :



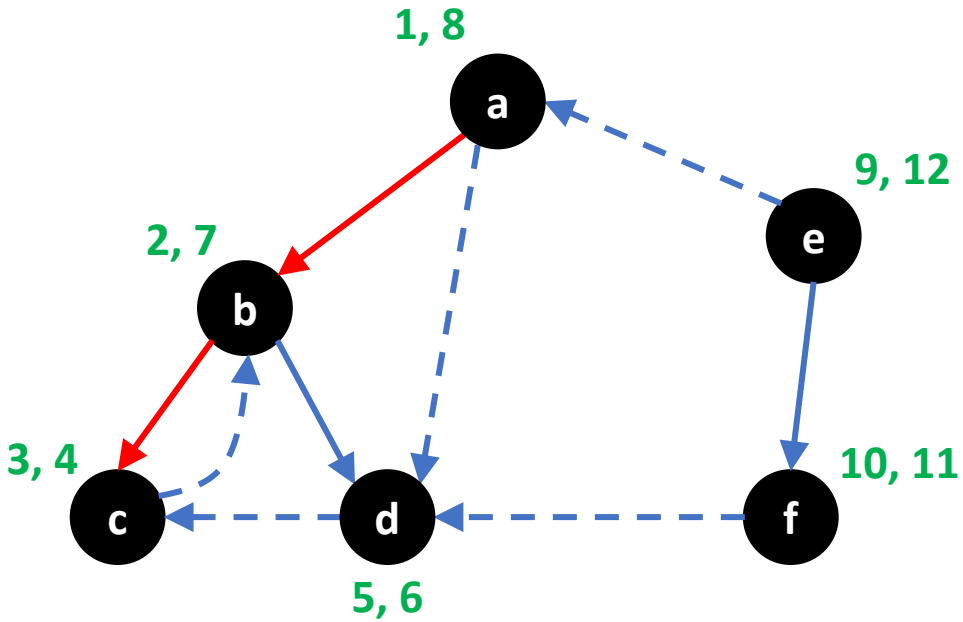
Colors of nodes:

B	B	B	B	B	B
a	**b**	**c**	**d**	**e**	**f**

a is **ancestor** of c

c is **descendant** of a

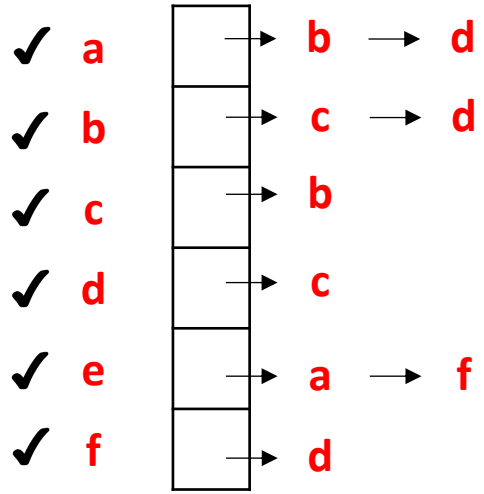
DFS(G)



1

d[a]

Adj List of G :



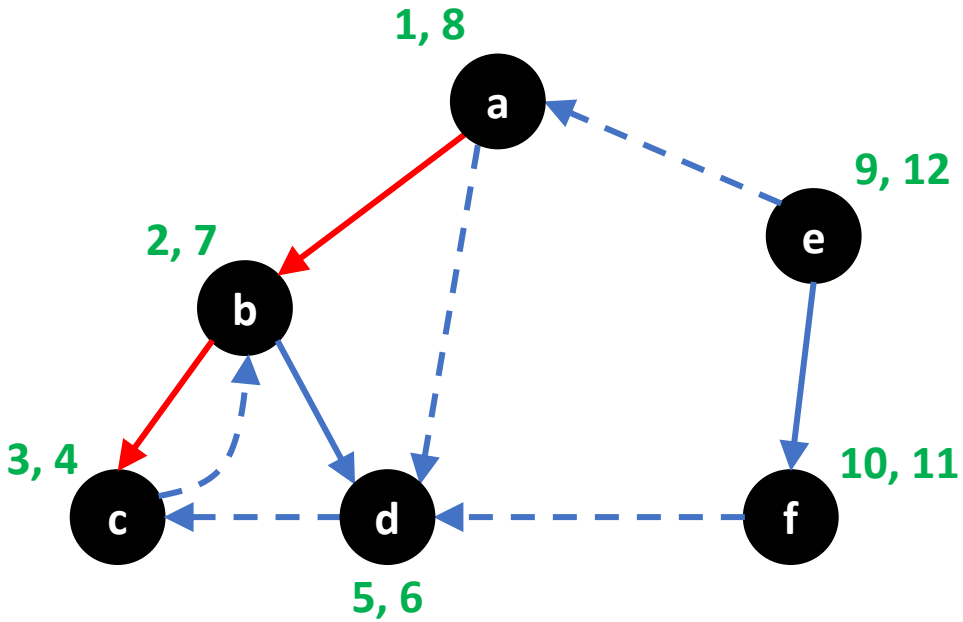
Colors of nodes:

B	B	B	B	B	B
a	**b**	**c**	**d**	**e**	**f**

a is **ancestor** of c

c is **descendant** of a

DFS(G)



1

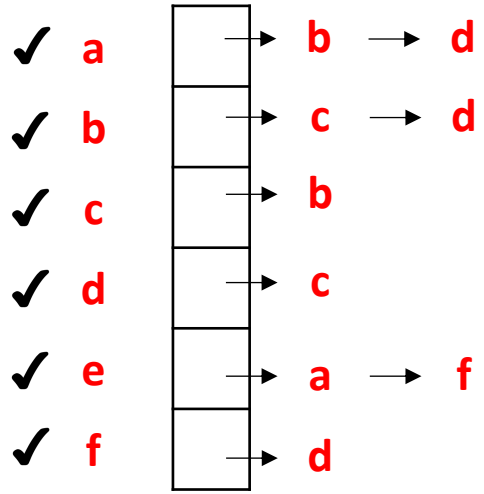
d[a] < d[c]

3

a is **ancestor** of c

c is **descendant** of a

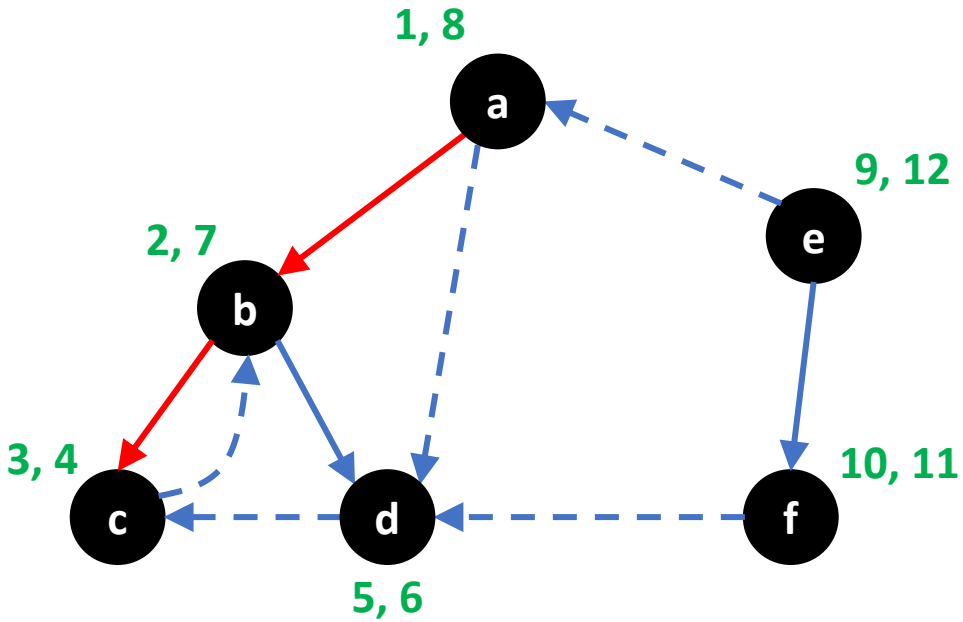
Adj List of G :



Colors of nodes:

B	B	B	B	B	B
a	**b**	**c**	**d**	**e**	**f**

DFS(G)



1

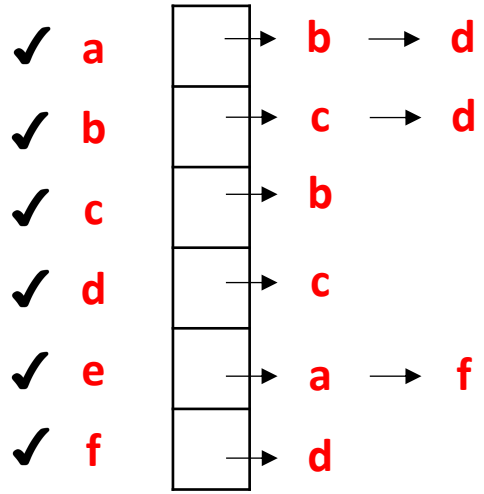
d[a] < d[c] < f[c]

3 4

a is **ancestor** of c

c is **descendant** of a

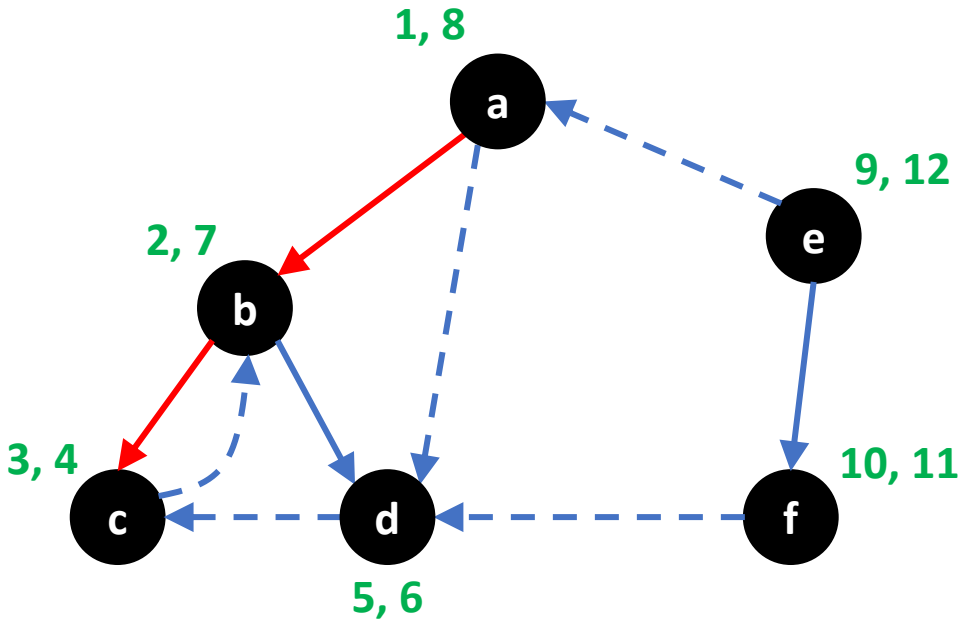
Adj List of G :



Colors of nodes:

B	B	B	B	B	B
a	b	c	d	e	f

DFS(G)



1 8

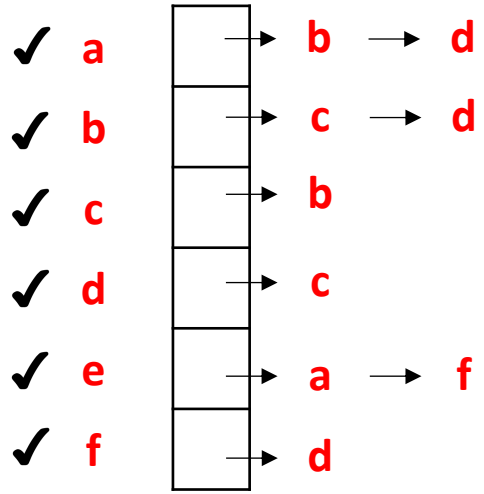
d[a] < d[c] < f[c] < f[a]

3 4

a is **ancestor** of c

c is **descendant** of a

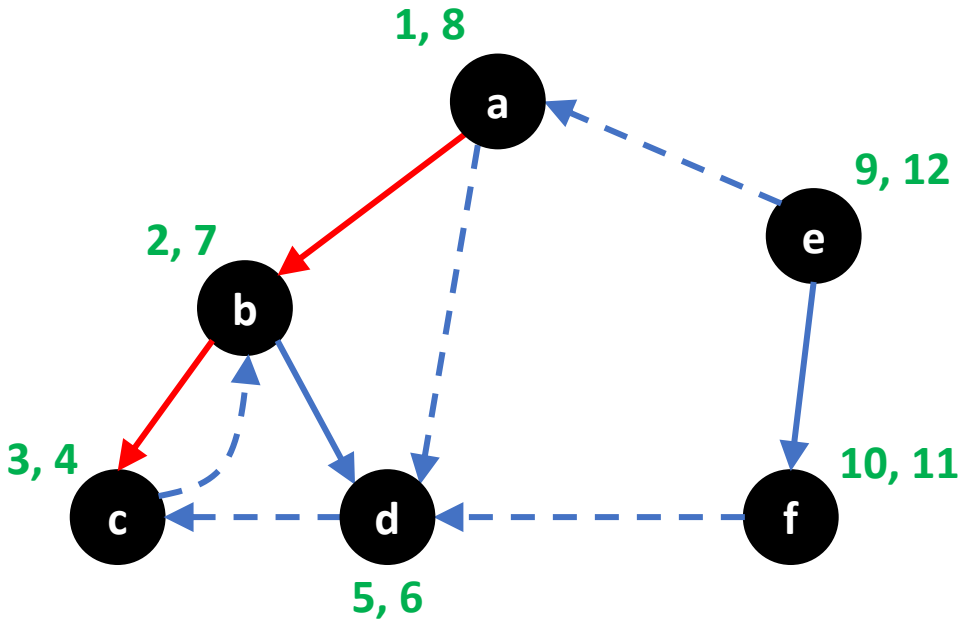
Adj List of G :



Colors of nodes:

B	B	B	B	B	B
a	**b**	**c**	**d**	**e**	**f**

DFS(G)



1 8

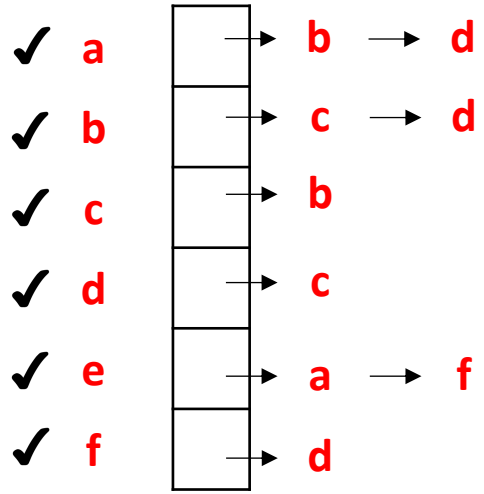
d[a] < d[c] < f[c] < f[a]

3 4

a is **ancestor** of c

c is **descendant** of a

Adj List of G :

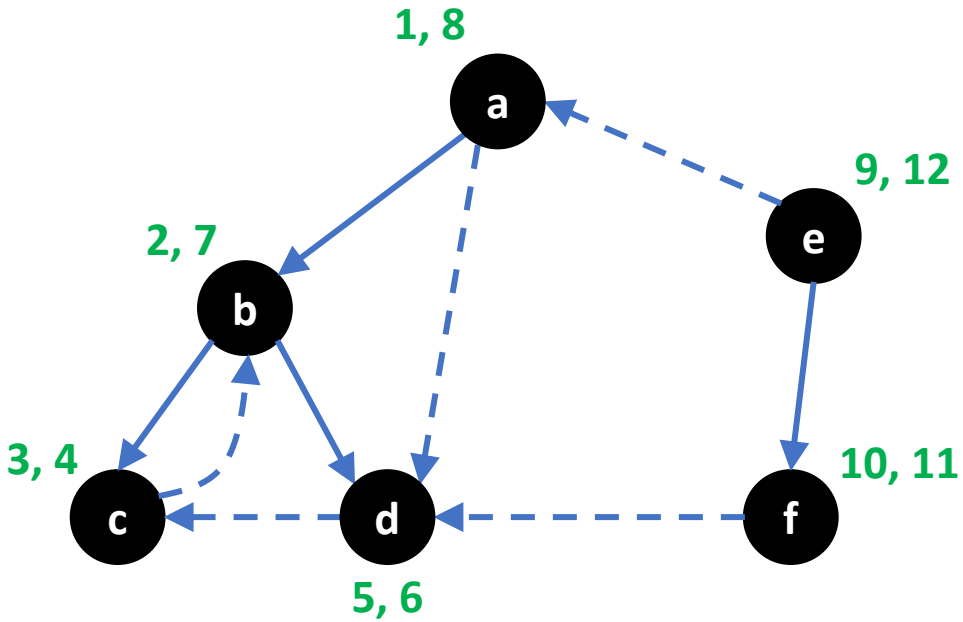


Colors of nodes:

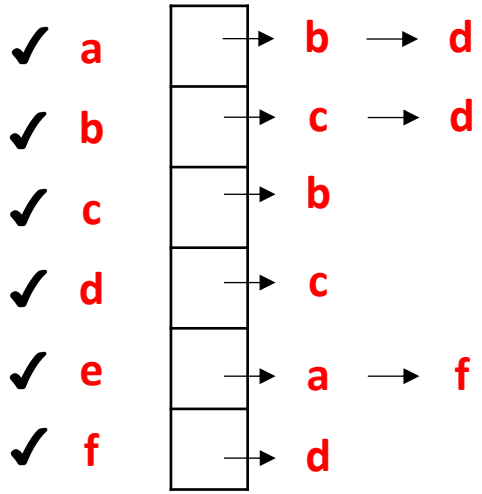
B	B	B	B	B	B
a	**b**	**c**	**d**	**e**	**f**

DFS(G)

Edge Classification by DFS



Adj List of G :

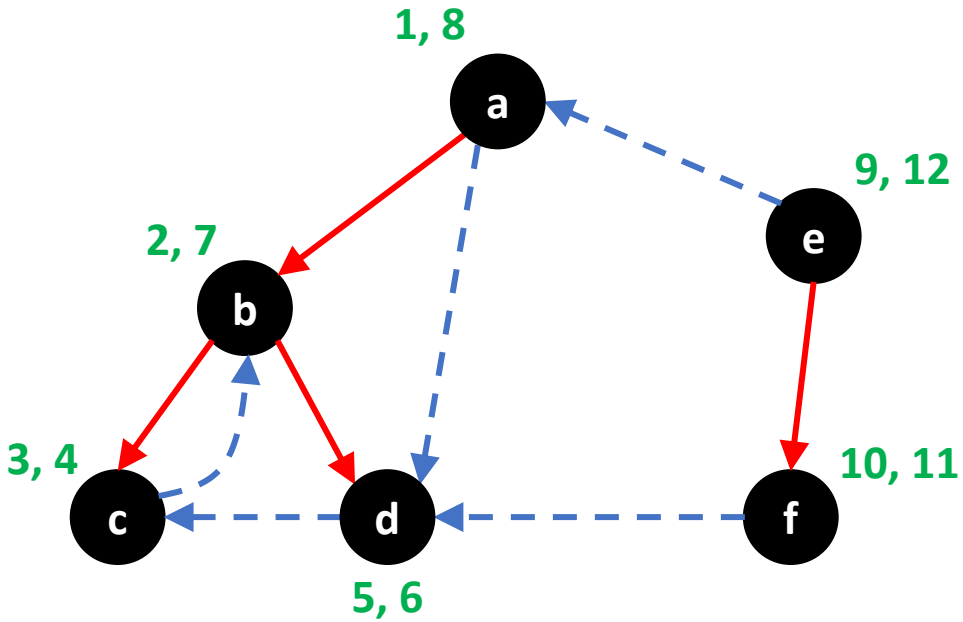


Colors of nodes:

B	B	B	B	B	B
a	**b**	**c**	**d**	**e**	**f**

DFS(G)

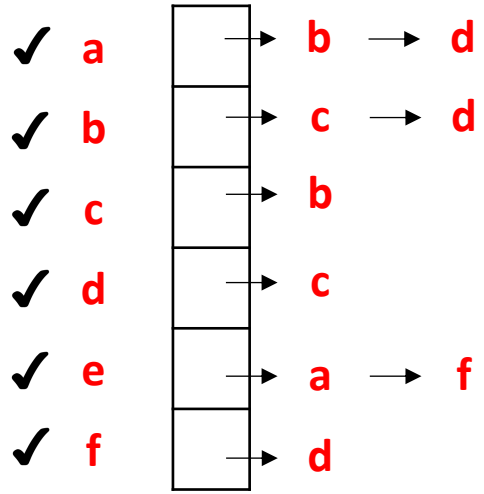
Edge Classification by DFS



An edge (u, v) of G is a

Tree edge ⇔ u is the **parent** of v in the DFS forest

Adj List of G :

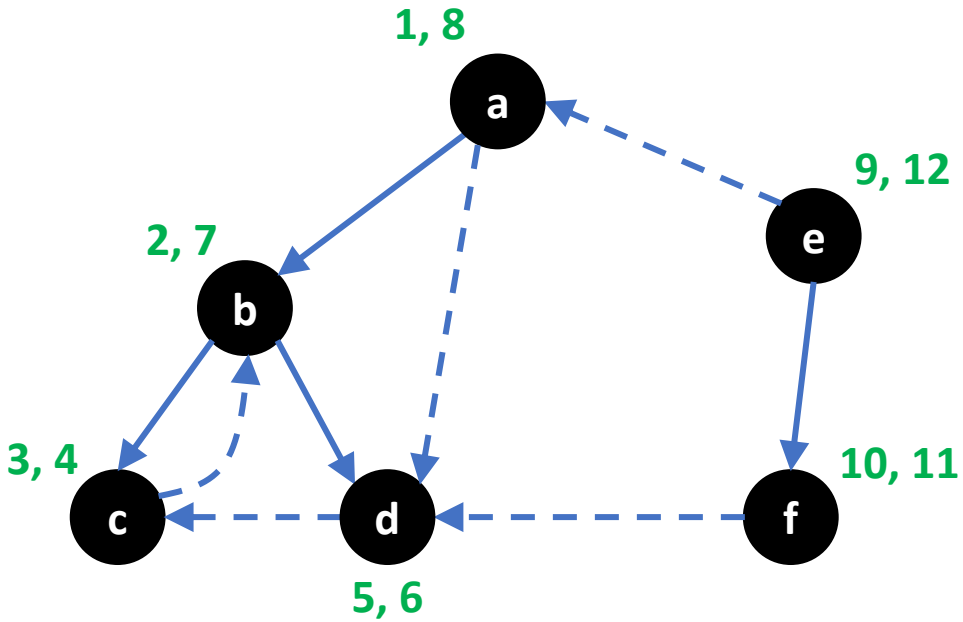


Colors of nodes:

B	B	B	B	B	B
a	**b**	**c**	**d**	**e**	**f**

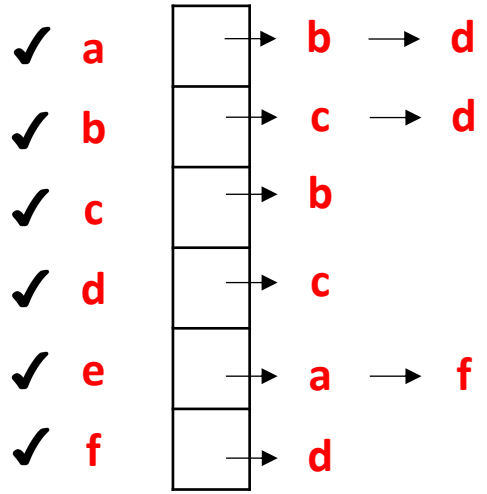
DFS(G)

Edge Classification by DFS



A **non-tree** edge (u, v) of G is a

Adj List of G :

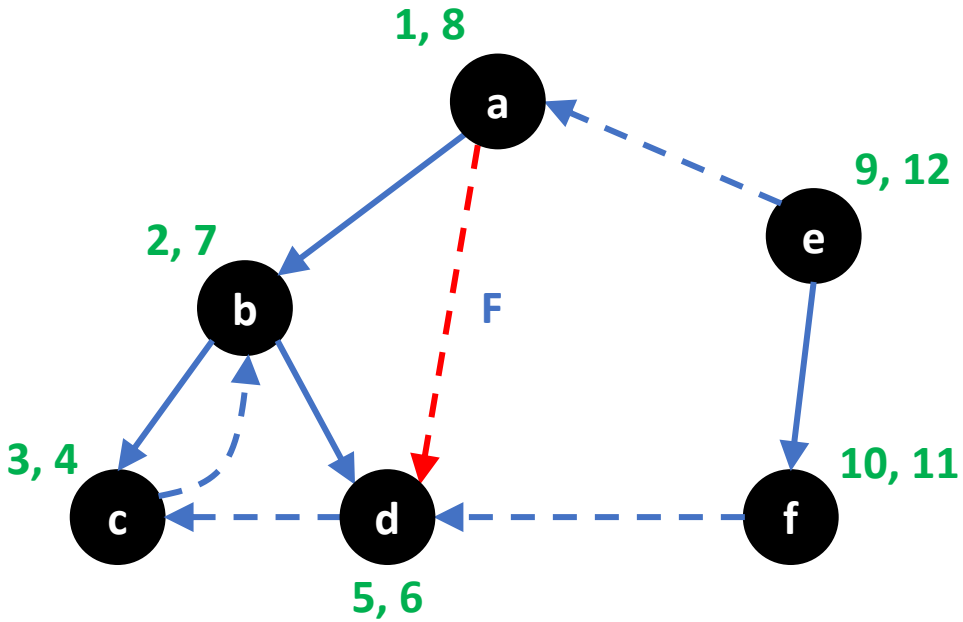


Colors of nodes:

B	B	B	B	B	B
a	**b**	**c**	**d**	**e**	**f**

DFS(G)

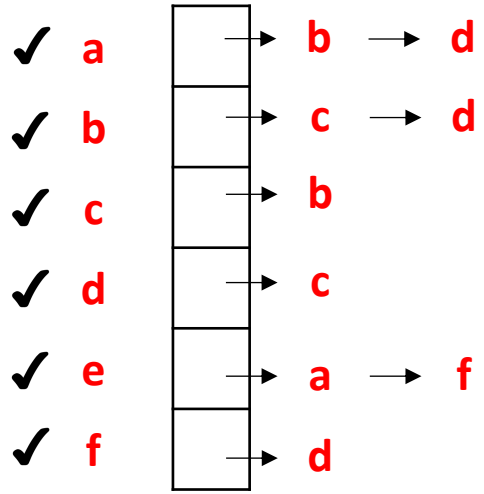
Edge Classification by DFS



A **non-tree** edge (u, v) of G is a

Forward edge ⇔ u is an **ancestor** of v in the DFS forest

Adj List of G :

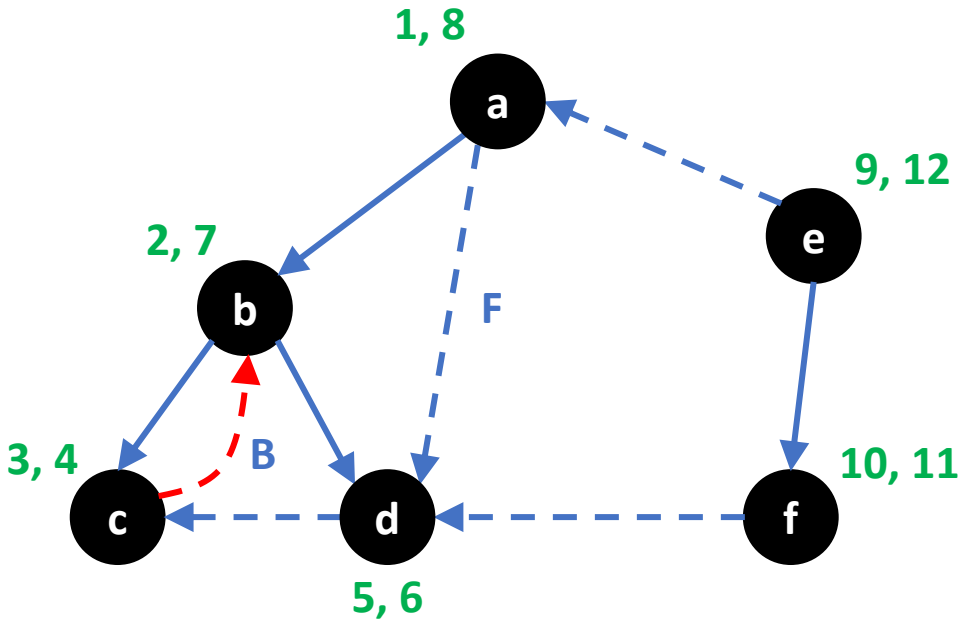


Colors of nodes:

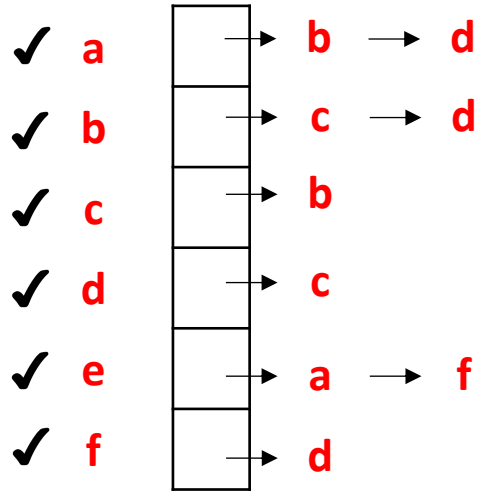
a	b	c	d	e	f
B	B	B	B	B	B

DFS(G)

Edge Classification by DFS



Adj List of G :



A **non-tree** edge (u, v) of G is a

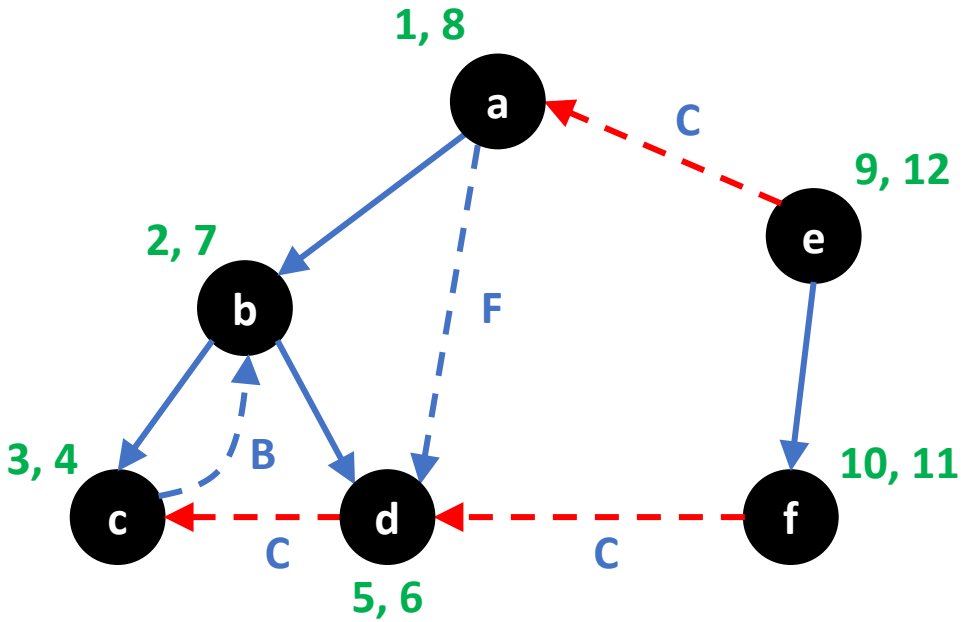
Back edge ⇔ u is a **descendent** of v in the DFS forest

Colors of nodes:

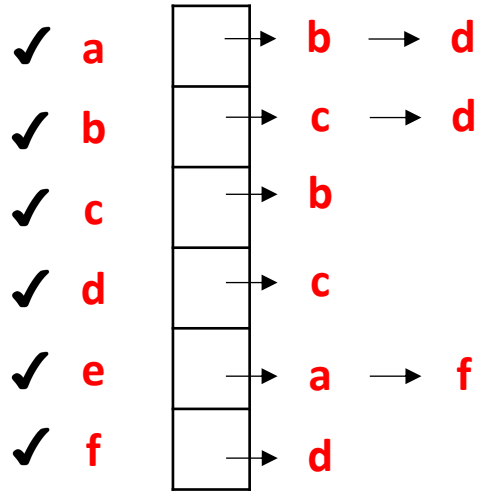
a	b	c	d	e	f
B	B	B	B	B	B

DFS(G)

Edge Classification by DFS



Adj List of G :



Colors of nodes:

a	b	c	d	e	f
B	B	B	B	B	B

A **non-tree** edge (u, v) of G is a

Cross edge ⇔ u is **neither ancestor nor descendent** of v in the DFS forest
⇔ not a forward or back edge

Edge Classification by DFS

A DFS of a directed graph G = (V, E) classifies the edges of G as follows:

Edge Classification by DFS

A DFS of a directed graph G = (V, E) classifies the edges of G as follows:

(u,v) ∈ E is a :

1. **Tree edge**

Edge Classification by DFS

A DFS of a directed graph G = (V, E) classifies the edges of G as follows:

(u,v) ∈ E is a :

1. **Tree edge** ⇔ u is the **parent** of v in the DFS forest

Edge Classification by DFS

A DFS of a directed graph G = (V, E) classifies the edges of G as follows:

(u,v) ∈ E is a :

1. **Tree edge** ⇔ u is the **parent** of v in the DFS forest

Non-tree edges

Edge Classification by DFS

A DFS of a directed graph G = (V, E) classifies the edges of G as follows:

(u,v) ∈ E is a :

1. **Tree edge** ⇔ u is the **parent** of v in the DFS forest
2. **Forward edge** ⇔ u is an **ancestor** of v in the DFS forest

Non-tree edges

Edge Classification by DFS

A DFS of a directed graph G = (V, E) classifies the edges of G as follows:

(u,v) ∈ E is a :

1. **Tree edge** ⇔ u is the **parent** of v in the DFS forest
2. **Forward edge** ⇔ u is an **ancestor** of v in the DFS forest
3. **Back edge** ⇔ u is a **descendant** of v in the DFS forest

Non-tree edges

Edge Classification by DFS

A DFS of a directed graph G = (V, E) classifies the edges of G as follows:

(u,v) ∈ E is a :

1. **Tree edge** ⇔ u is the **parent** of v in the DFS forest
2. **Forward edge** ⇔ u is an **ancestor** of v in the DFS forest
3. **Back edge** ⇔ u is a **descendant** of v in the DFS forest
4. **Cross edge** ⇔ u is **neither ancestor nor descendent** of v in the forest

Non-tree edges

Claim 1 :

u is an ancestor of v in a DFS of G

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ d[u]

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ d[u] < d[v]

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v]$

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Exploration of u

Exploration of v

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Exploration of u

Exploration of v

Claim 2 :

For any 2 nodes u and v,

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Exploration of u

Exploration of v

Claim 2 :

For any 2 nodes u and v, we CANNOT have d[u]

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Exploration of u

Exploration of v

Claim 2 :

For any 2 nodes u and v, we CANNOT have $d[u] < d[v]$

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Exploration of u

Exploration of v

Claim 2 :

For any 2 nodes u and v, we CANNOT have $d[u] < d[v] < f[u]$

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Exploration of u

Exploration of v

Claim 2 :

For any 2 nodes u and v, we CANNOT have $d[u] < d[v] < f[u] < f[v]$

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Exploration of u

Exploration of v

Claim 2 :

For any 2 nodes u and v, we CANNOT have $d[u] < d[v] < f[u] < f[v]$

Exploration of v

Exploration of u

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Exploration of u

Exploration of v

Claim 2 :

For any 2 nodes u and v, we CANNOT have $d[u] < d[v] < f[u] < f[v]$

Exploration of v

Exploration of u

Claim 3 :

If $(u,v) \in E$

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Exploration of u

Exploration of v

Claim 2 :

For any 2 nodes u and v, we CANNOT have $d[u] < d[v] < f[u] < f[v]$

Exploration of v

Exploration of u

Claim 3 :

If $(u,v) \in E$ then

v is surely discovered **before** we finish exploring u.

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Exploration of u

Exploration of v

Claim 2 :

For any 2 nodes u and v, we CANNOT have $d[u] < d[v] < f[u] < f[v]$

Exploration of v

Exploration of u

Claim 3 :

If $(u,v) \in E$ then d[v]

v is surely discovered **before** we finish exploring u.

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Exploration of u

Exploration of v

Claim 2 :

For any 2 nodes u and v, we CANNOT have $d[u] < d[v] < f[u] < f[v]$

Exploration of v

Exploration of u

Claim 3 :

If $(u,v) \in E$ then $d[v] < f[u]$

v is surely discovered **before** we finish exploring u.

Claim 1 :

u is an ancestor of v in a DFS of G $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Exploration of u

Exploration of v

Claim 2 :

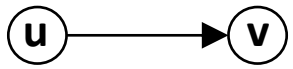
For any 2 nodes u and v, we CANNOT have $d[u] < d[v] < f[u] < f[v]$

Exploration of v

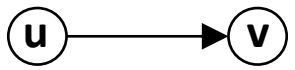
Exploration of u

Claim 3 :

If $(u,v) \in E$ then $d[v] < f[u]$

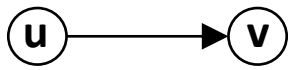
Claim: After a DFS of a directed graph, 

$\in E$ is of type:

Claim: After a DFS of a directed graph, 

$\in E$ is of type:

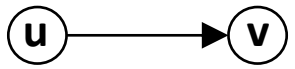
Tree or Forward edge $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Claim: After a DFS of a directed graph, 

$\in E$ is of type:

Tree or Forward edge $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

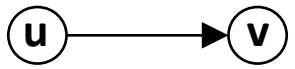
Back edge

Claim: After a DFS of a directed graph, 

$\in E$ is of type:

Tree or Forward edge $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Back edge $\Leftrightarrow$ $d[v] < d[u] < f[u] < f[v]$

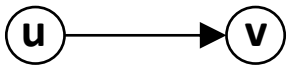
Claim: After a DFS of a directed graph, 

$\in E$ is of type:

Tree or Forward edge $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Back edge $\Leftrightarrow$ $d[v] < d[u] < f[u] < f[v]$

Cross edge

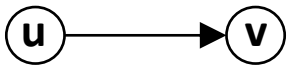
Claim: After a DFS of a directed graph, 

$\in E$ is of type:

Tree or Forward edge $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Back edge $\Leftrightarrow$ $d[v] < d[u] < f[u] < f[v]$

Cross edge $\Leftrightarrow$ $d[v] < f[v] < d[u] < f[u]$

Claim: After a DFS of a directed graph, 

$\in E$ is of type:

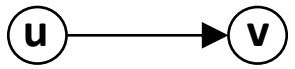
Tree or Forward edge $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Back edge $\Leftrightarrow$ $d[v] < d[u] < f[u] < f[v]$

Cross edge $\Leftrightarrow$ $d[v] < f[v] < d[u] < f[u]$

OR

$d[u] < f[u] < d[v] < f[v]$?

Claim: After a DFS of a directed graph, 

$\in E$ is of type:

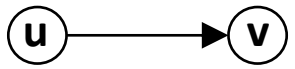
Tree or Forward edge $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

Back edge $\Leftrightarrow$ $d[v] < d[u] < f[u] < f[v]$

Cross edge $\Leftrightarrow$ $d[v] < f[v] < d[u] < f[u]$

OR

$d[u] < f[u] < d[v] < f[v]$?

Claim: After a DFS of a directed graph, 

$\in E$ is of type:

Tree or Forward edge $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

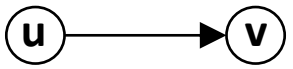
Back edge $\Leftrightarrow$ $d[v] < d[u] < f[u] < f[v]$

Cross edge $\Leftrightarrow$ $d[v] < f[v] < d[u] < f[u]$

OR

$d[u] < f[u] < d[v] < f[v]$

Impossible because of Claim 3 !

Claim: After a DFS of a directed graph, 

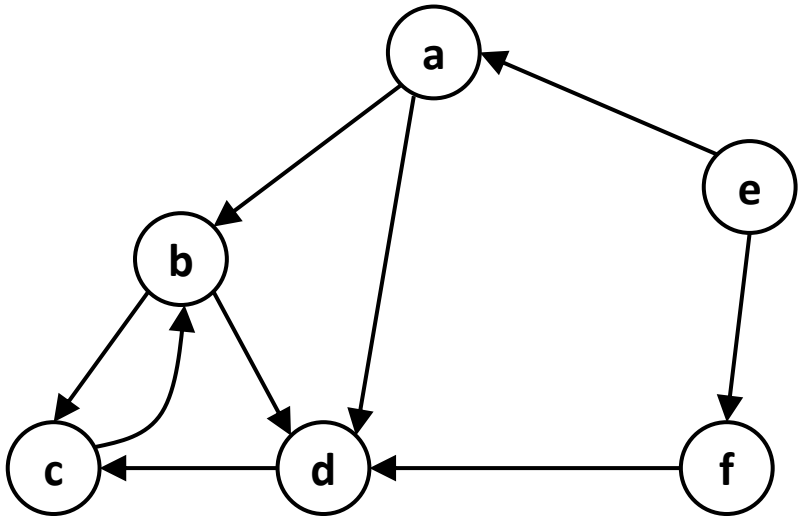
$\in E$ is of type:

Tree or Forward edge $\Leftrightarrow$ $d[u] < d[v] < f[v] < f[u]$

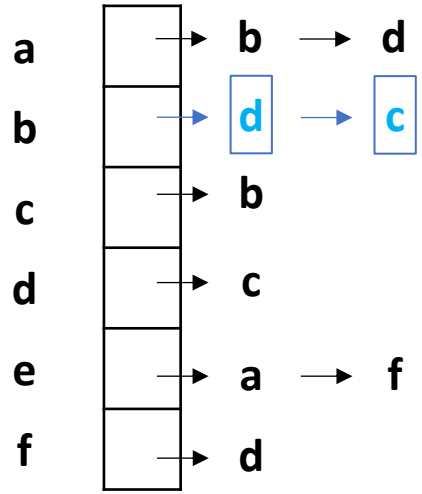
Back edge $\Leftrightarrow$ $d[v] < d[u] < f[u] < f[v]$

Cross edge $\Leftrightarrow$ $d[v] < f[v] < d[u] < f[u]$

DFS(G)



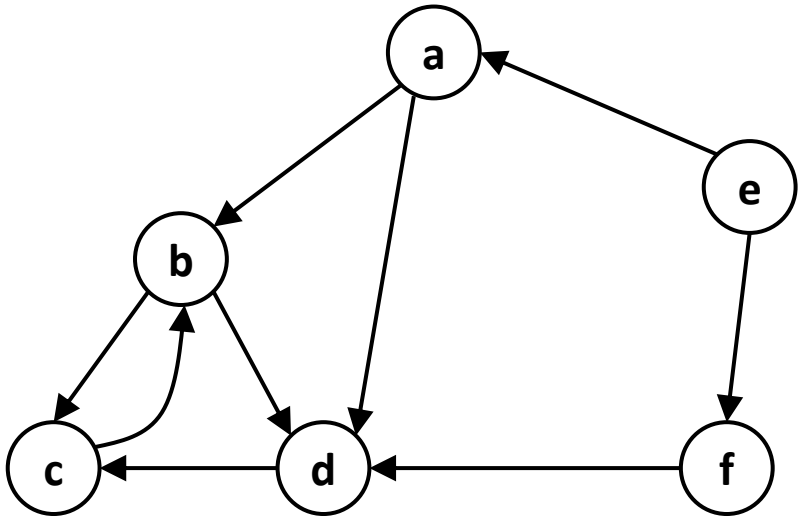
Adj List of G :



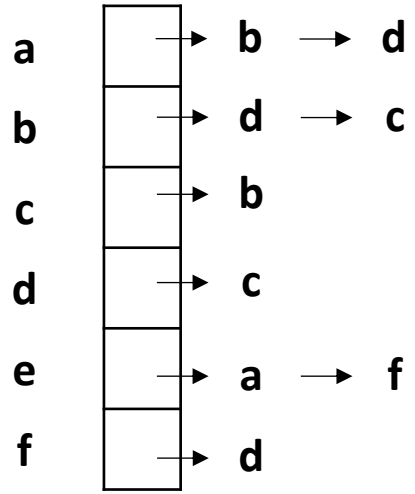
Colors of nodes:

e	a	b	c	d	f
W	W	W	W	W	W

DFS(G)



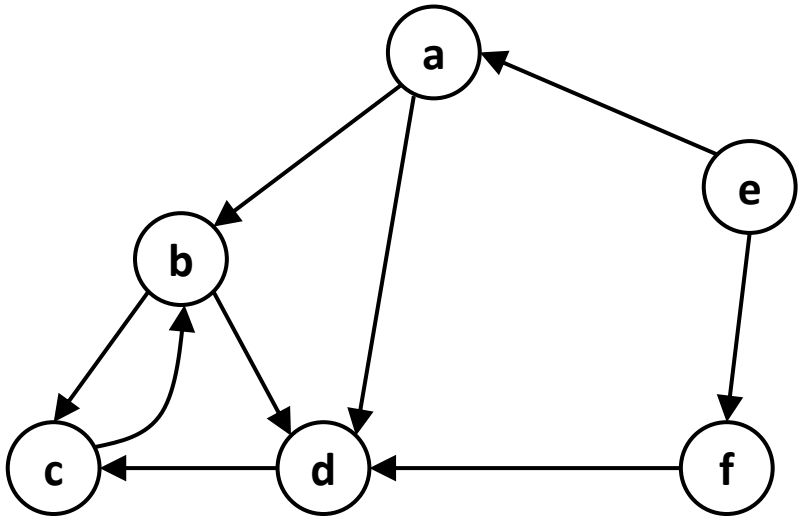
Adj List of G :



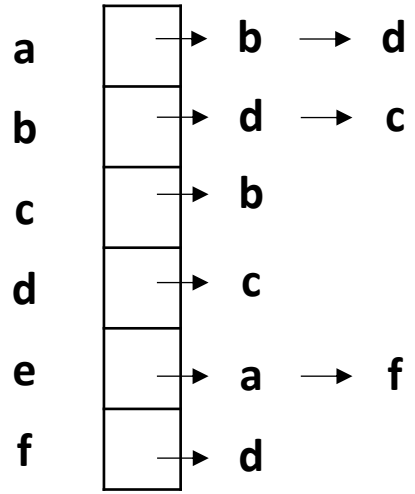
Colors of nodes:

e	a	b	c	d	f
W	W	W	W	W	W

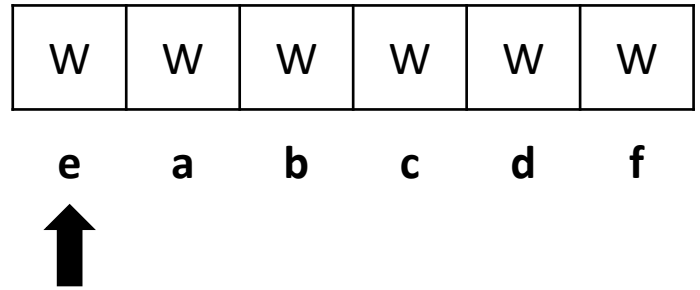
DFS(G)



Adj List of G :

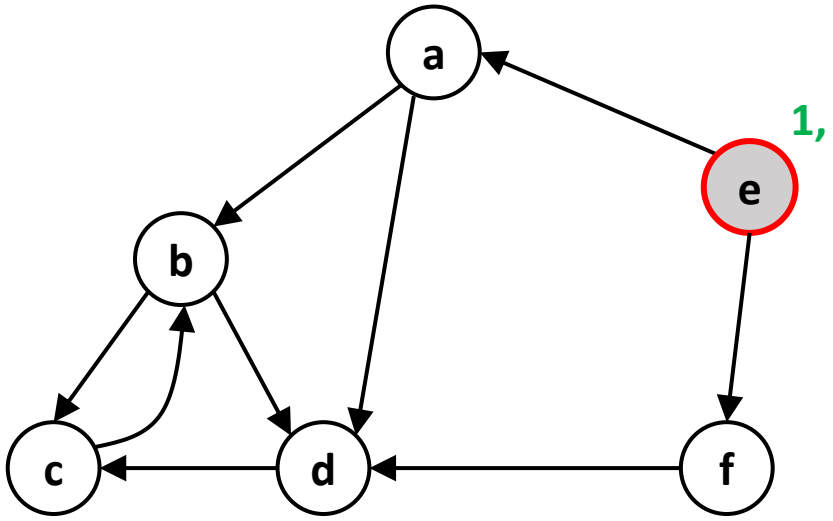


Colors of nodes:

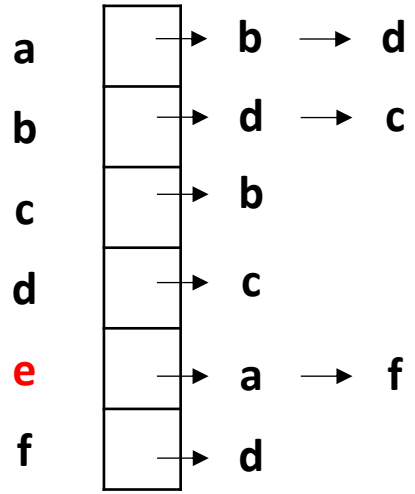


DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

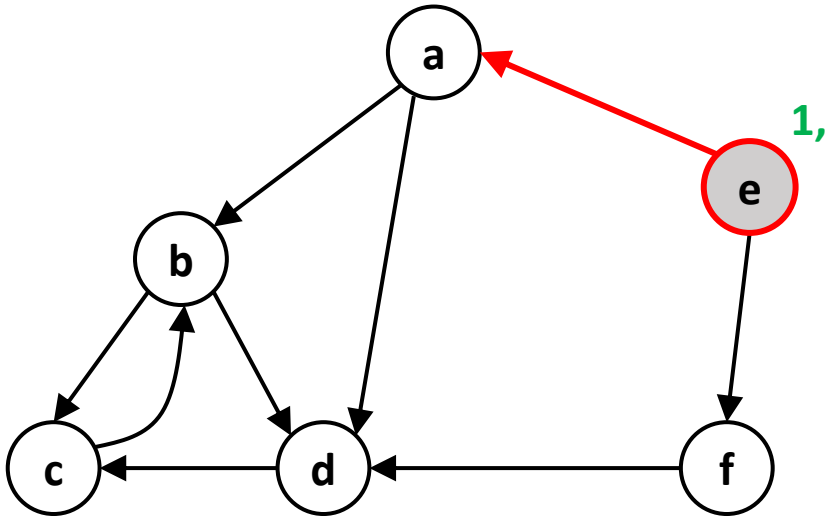


Colors of nodes:

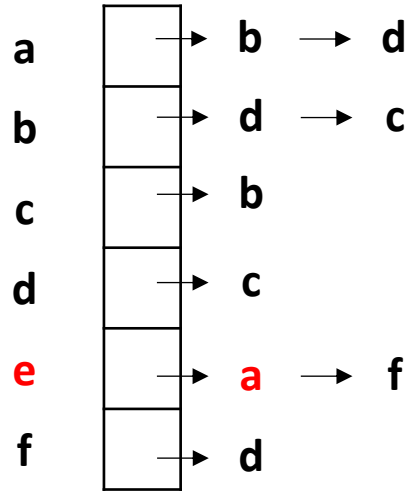
G	W	W	W	W	W
e	a	b	c	d	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

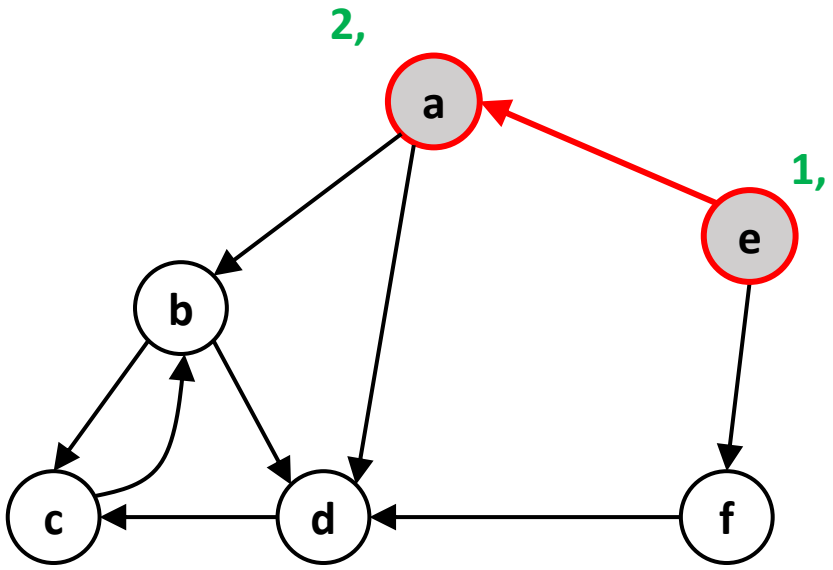


Colors of nodes:

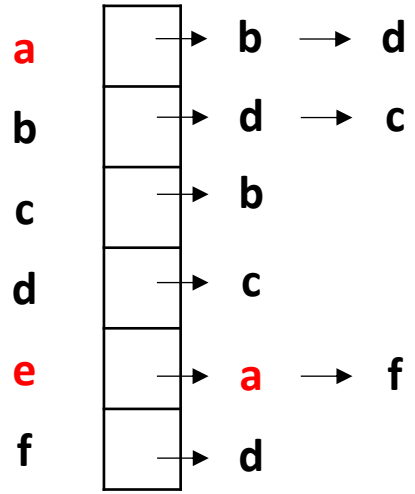
G	W	W	W	W	W
e	a	b	c	d	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

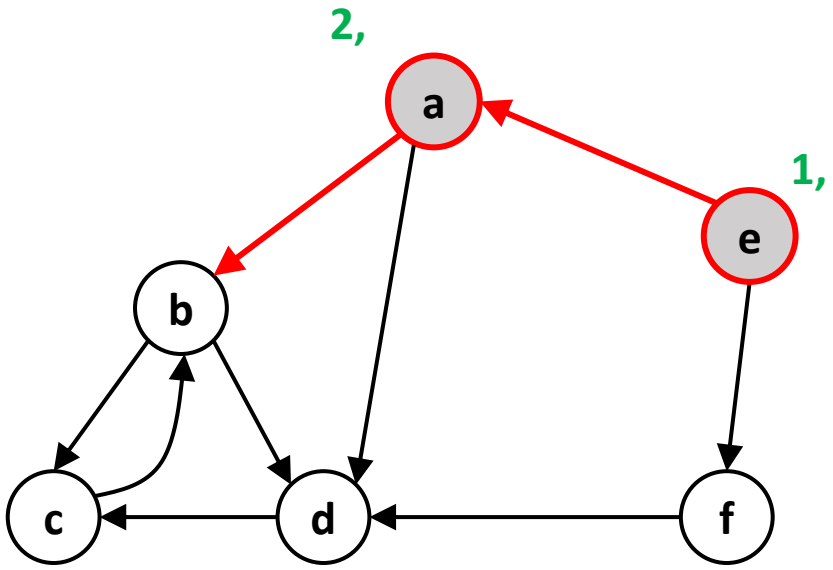


Colors of nodes:

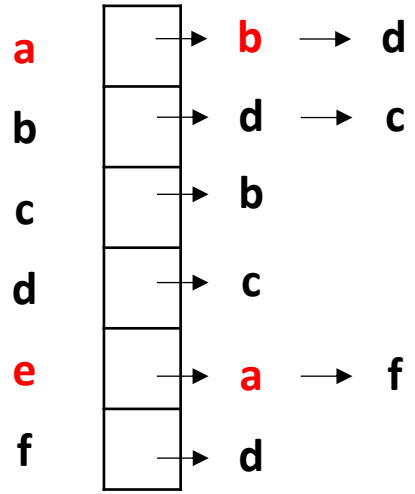
G	G	W	W	W	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

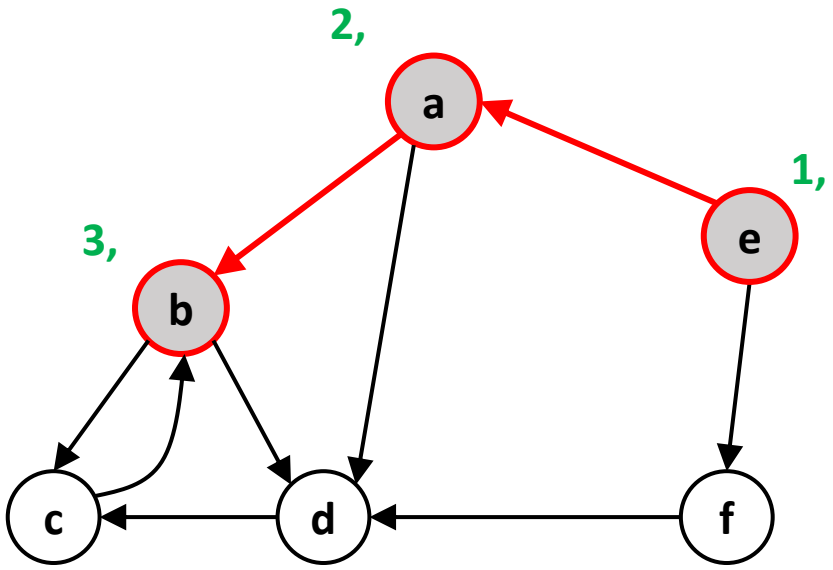


Colors of nodes:

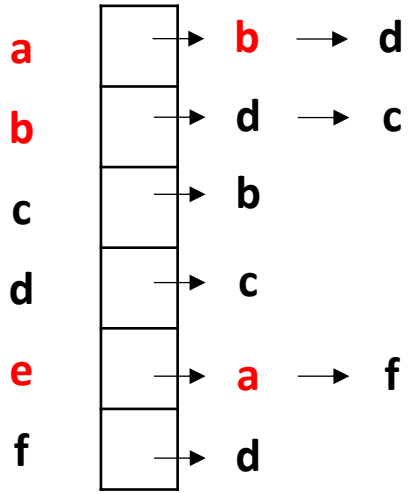
G	G	W	W	W	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

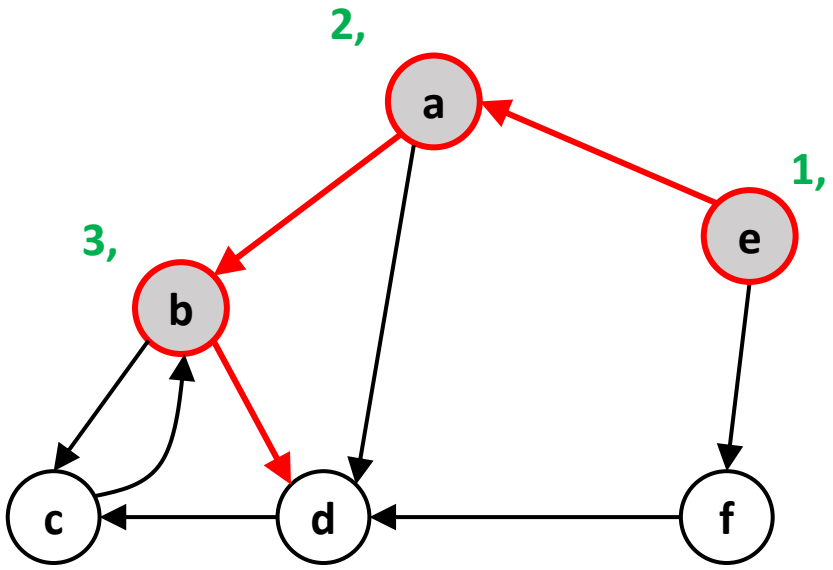


Colors of nodes:

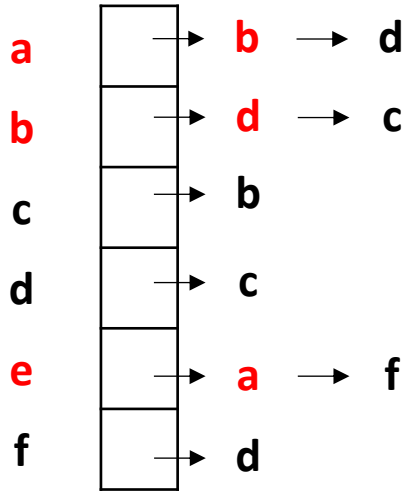
G	G	G	W	W	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

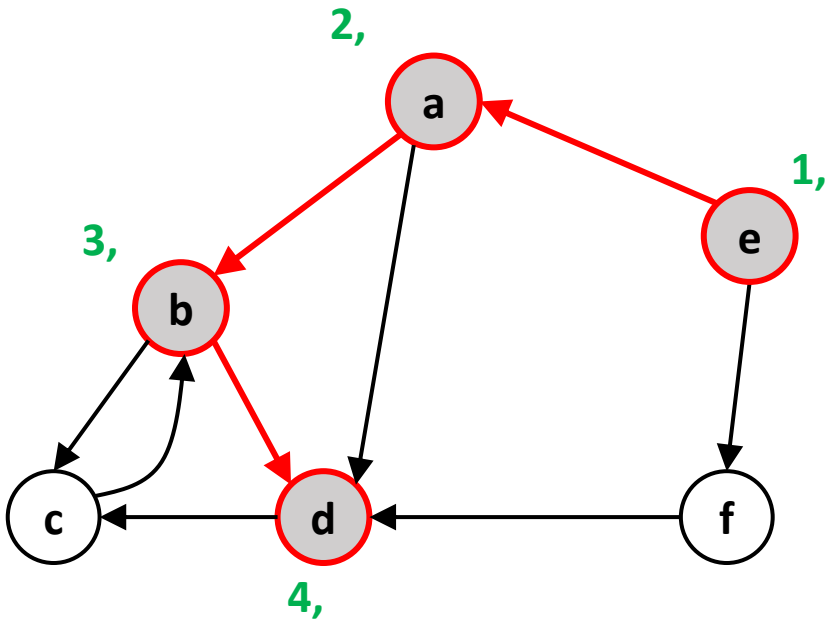


Colors of nodes:

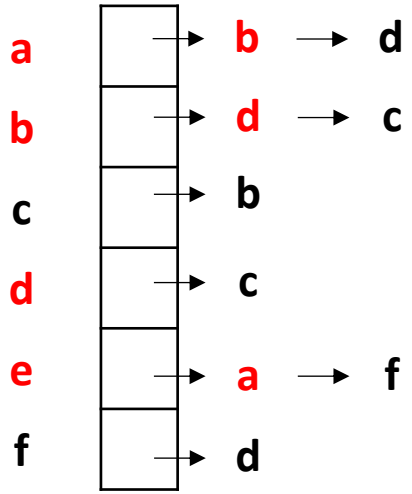
G	G	G	W	W	W
e	a	b	c	d	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

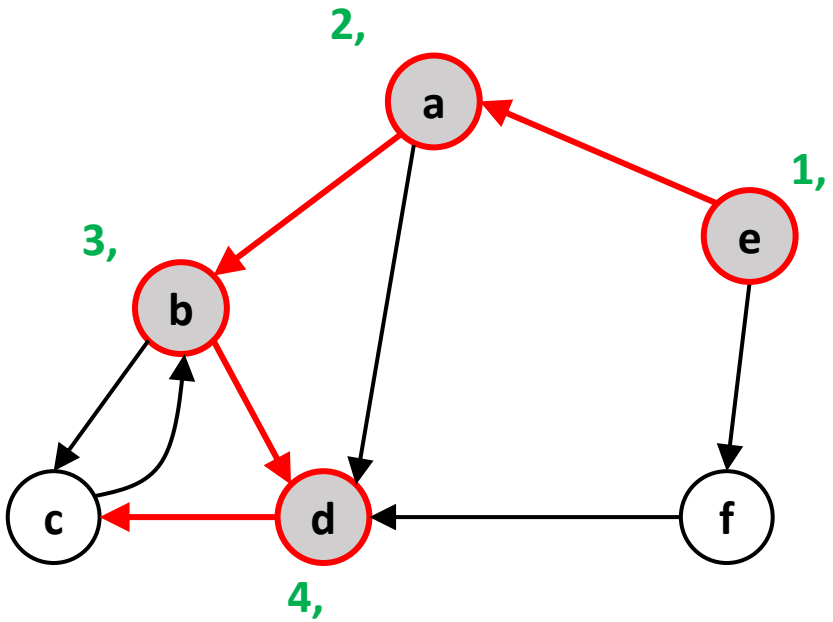


Colors of nodes:

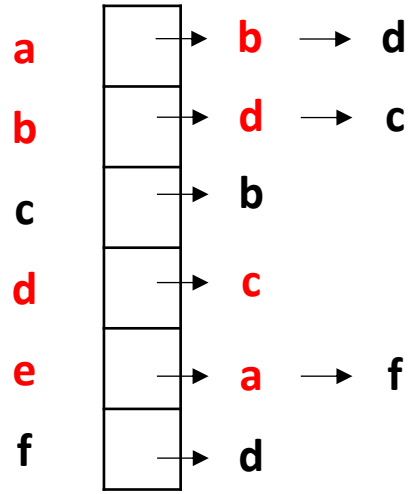
G	G	G	W	G	W
e	a	b	c	d	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

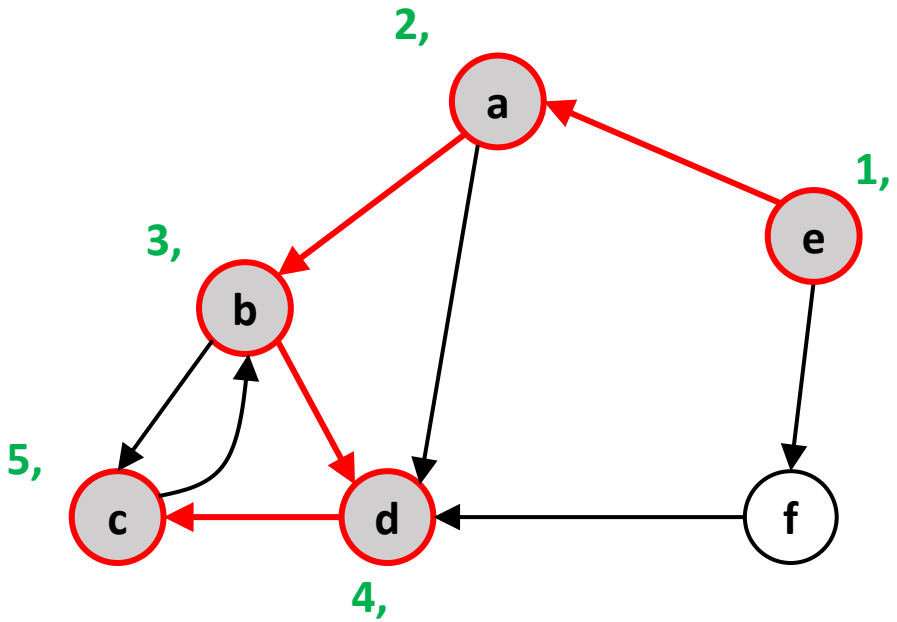


Colors of nodes:

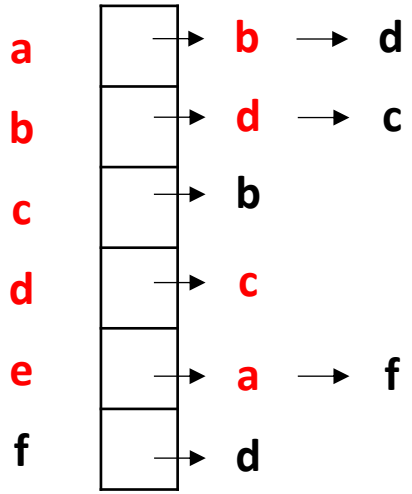
G	G	G	W	G	W
e	a	b	c	d	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

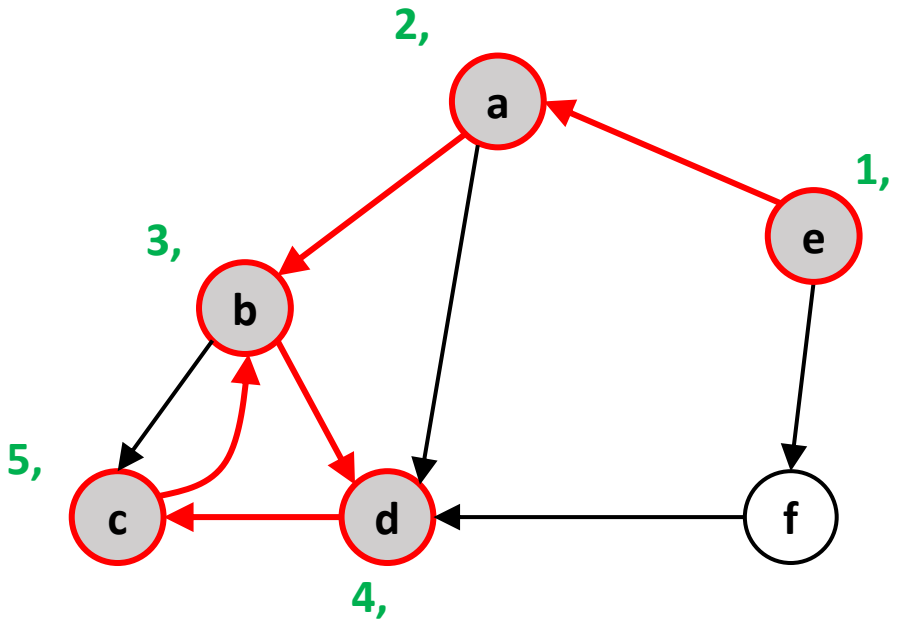


Colors of nodes:

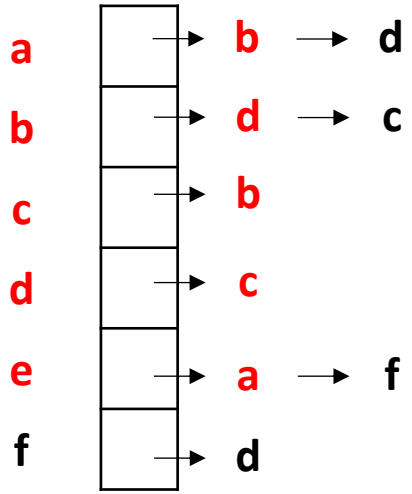
G	G	G	G	G	W
e	a	b	c	d	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

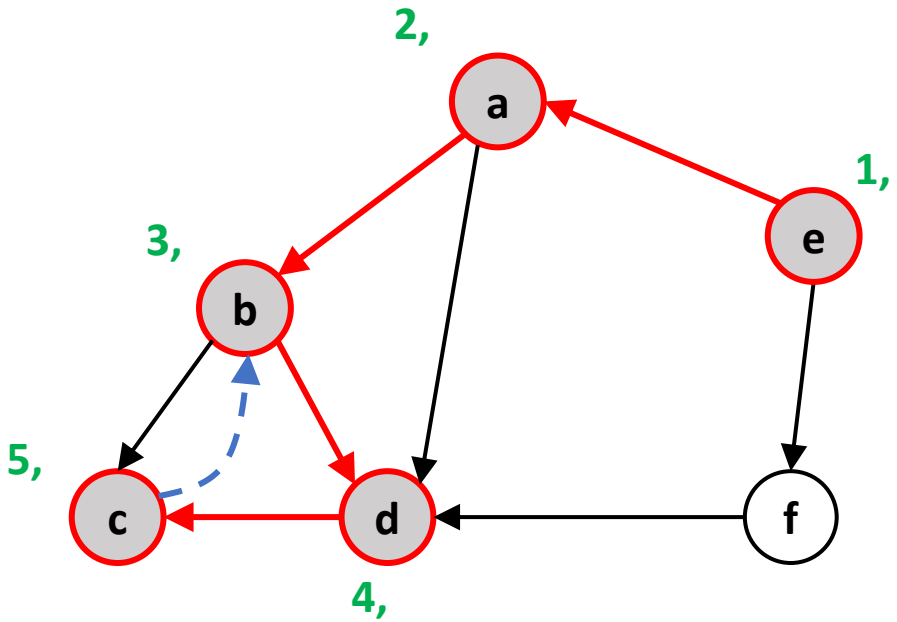


Colors of nodes:

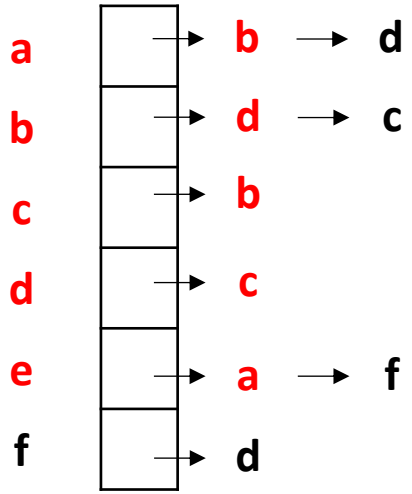
G	G	G	G	G	W
e	a	b	c	d	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

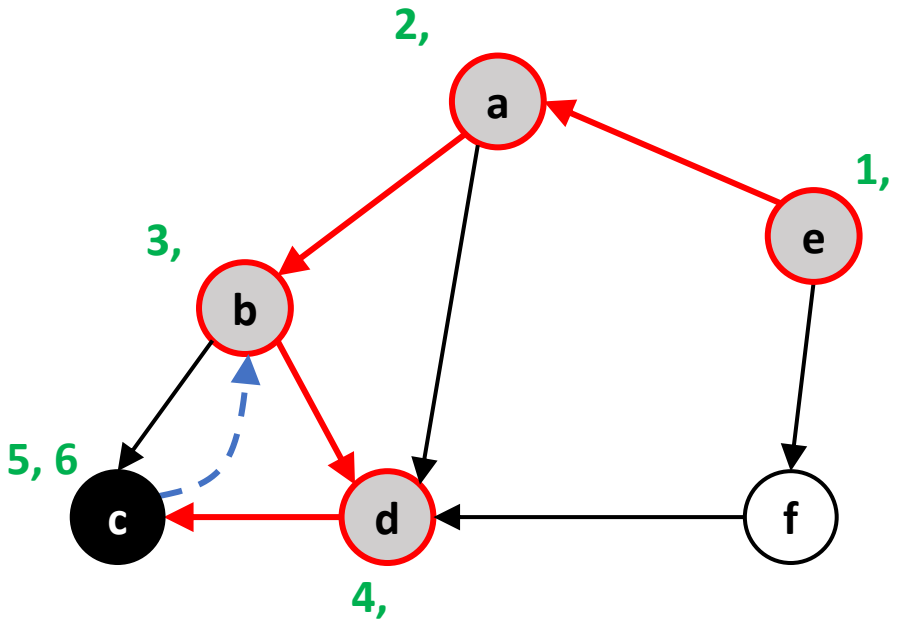


Colors of nodes:

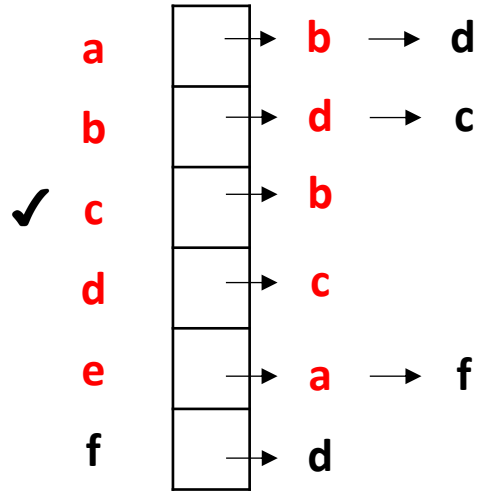
G	G	G	G	G	W
e	a	b	c	d	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

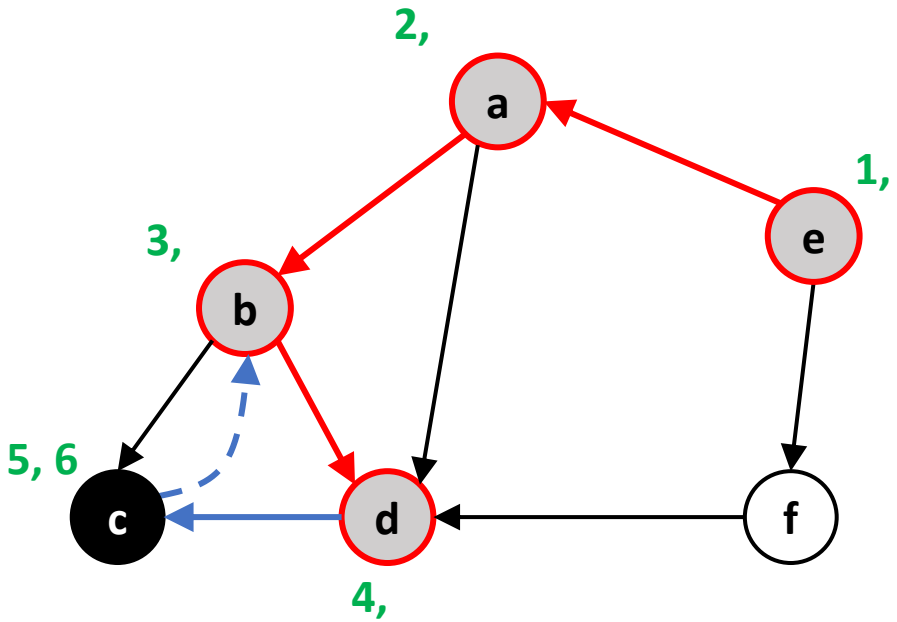


Colors of nodes:

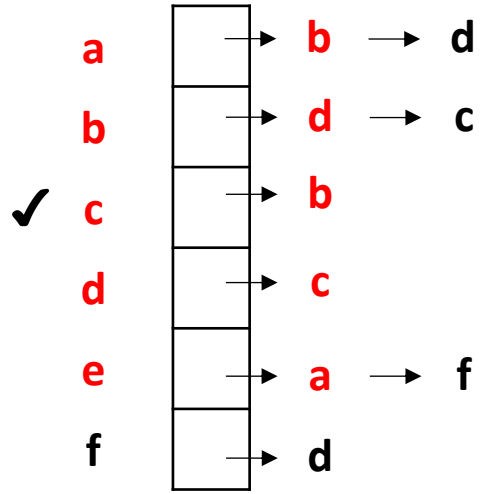
G	G	G	B	G	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

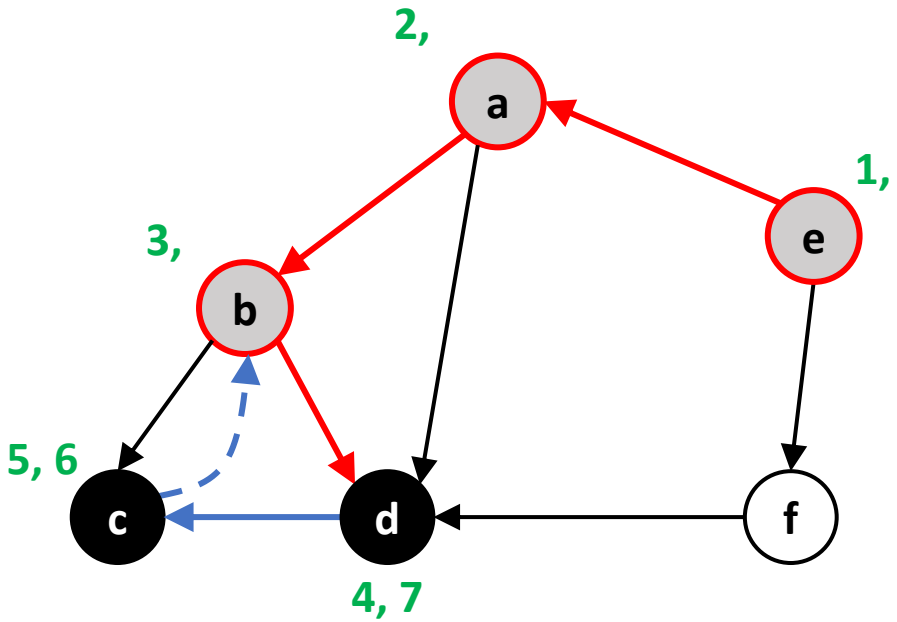


Colors of nodes:

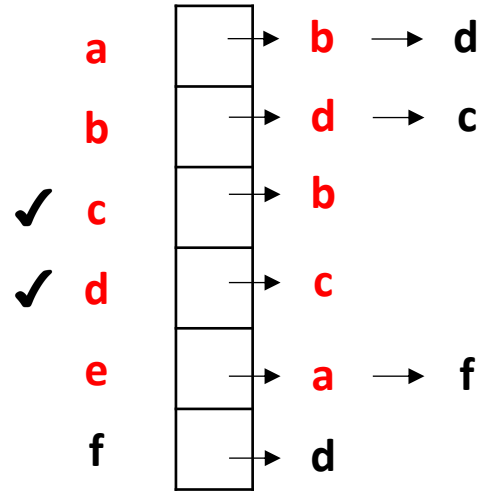
G	G	G	B	G	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

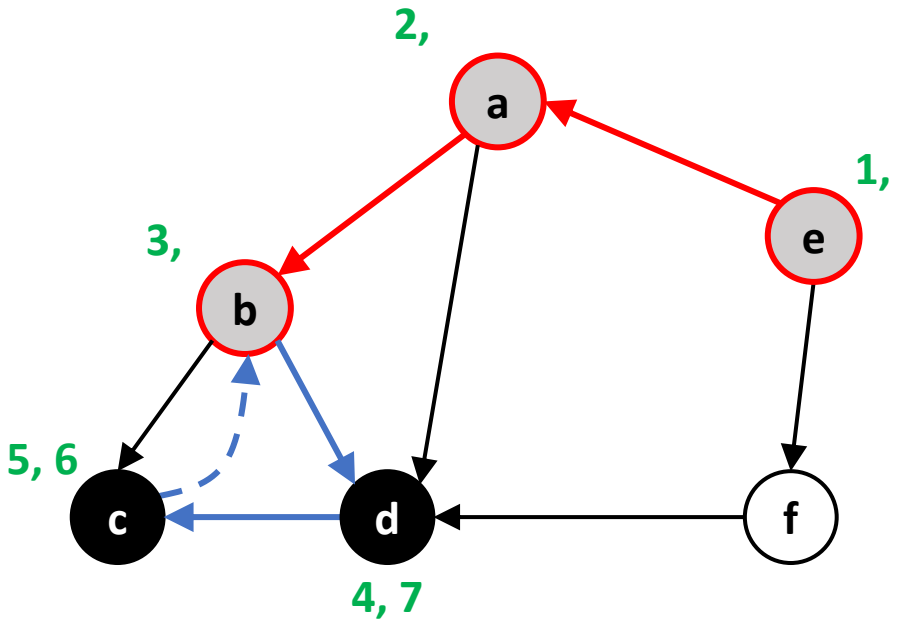


Colors of nodes:

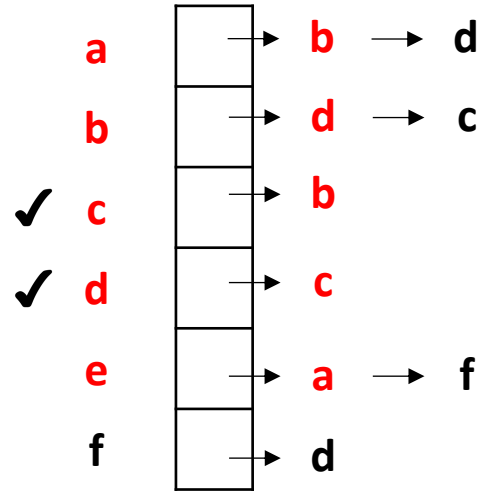
G	G	G	B	B	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

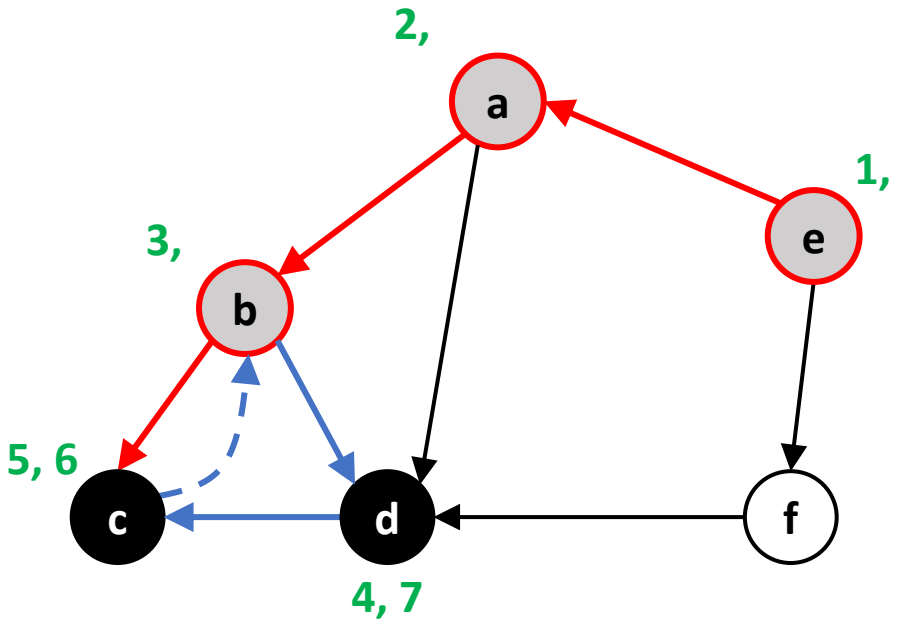


Colors of nodes:

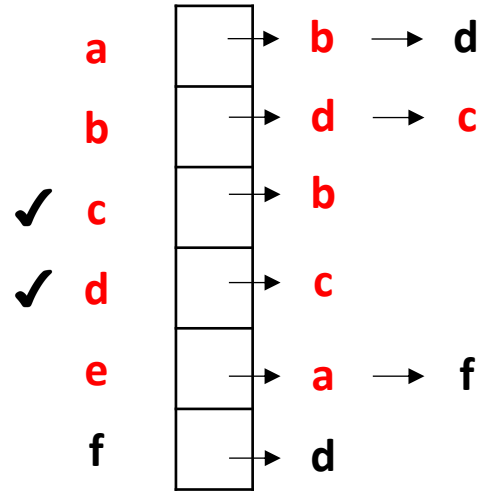
G	G	G	B	B	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

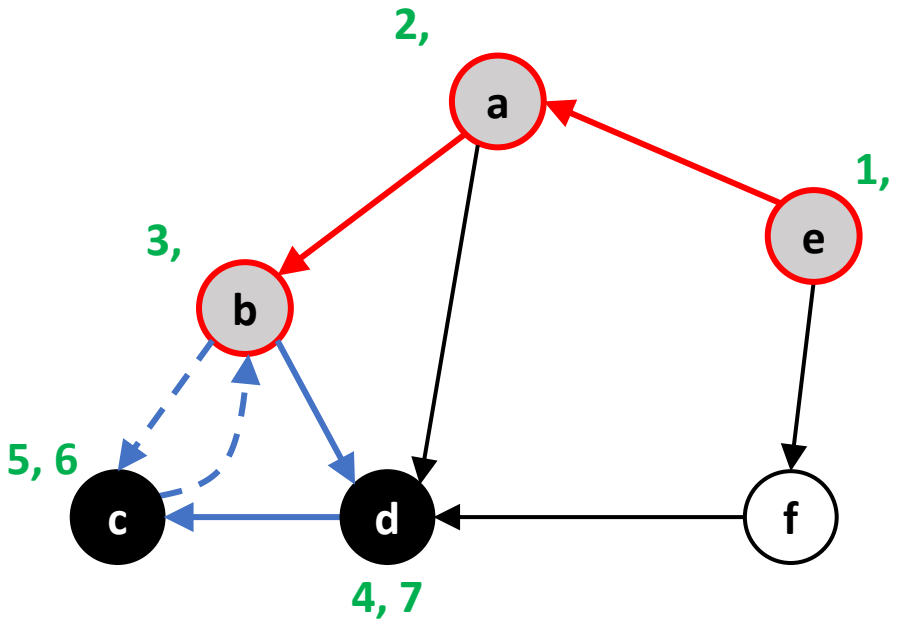


Colors of nodes:

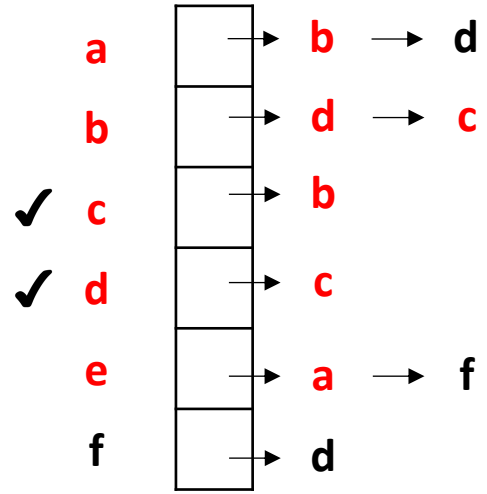
G	G	G	B	B	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

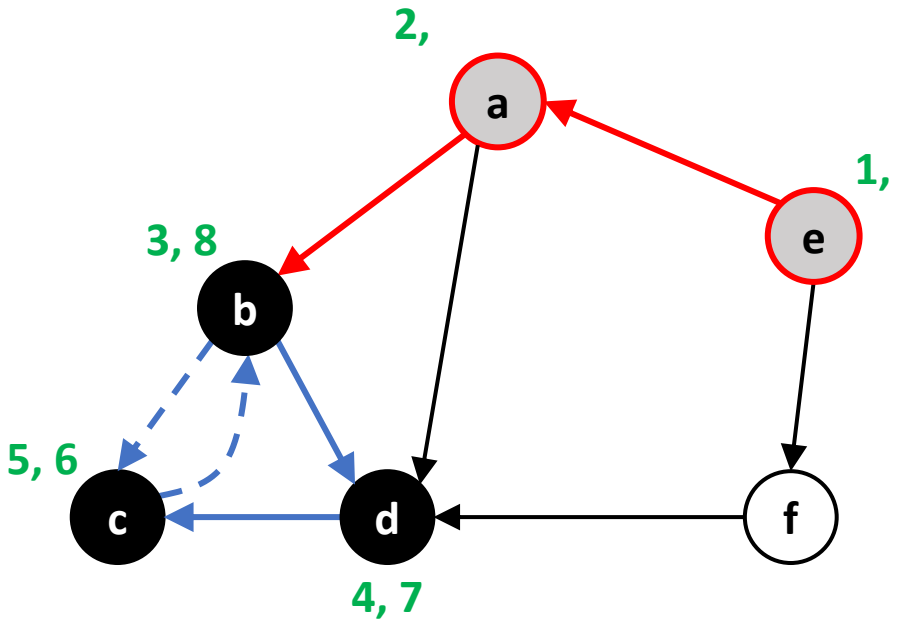


Colors of nodes:

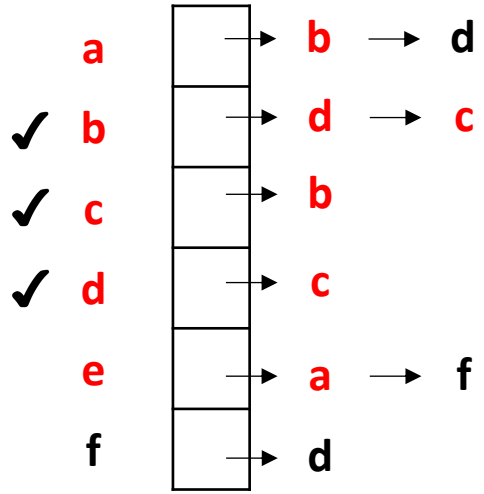
G	G	G	B	B	W
e	a	b	c	d	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

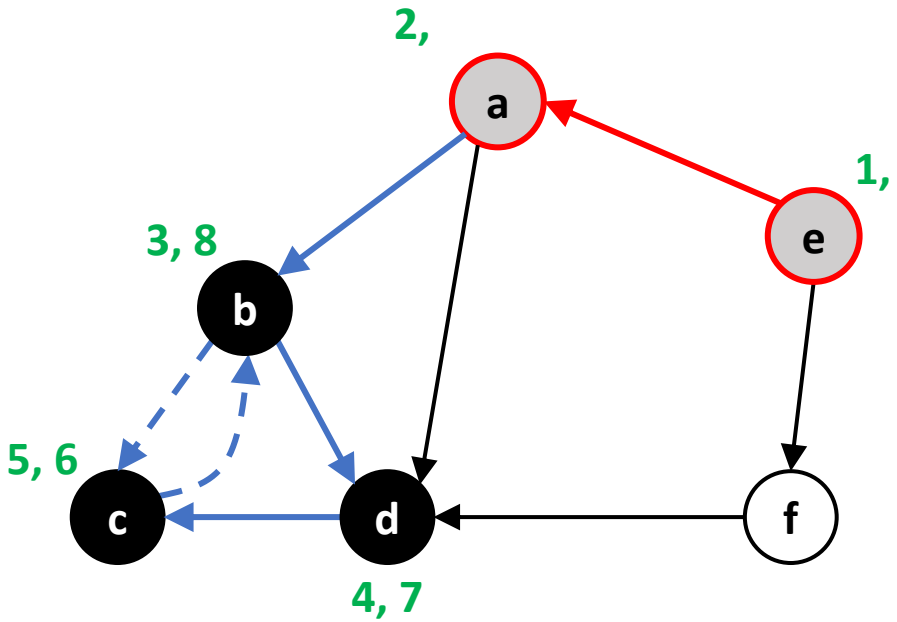


Colors of nodes:

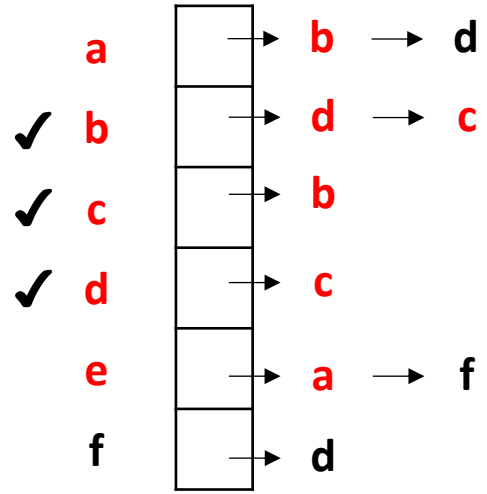
G	G	B	B	B	W
e	a	b	c	d	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

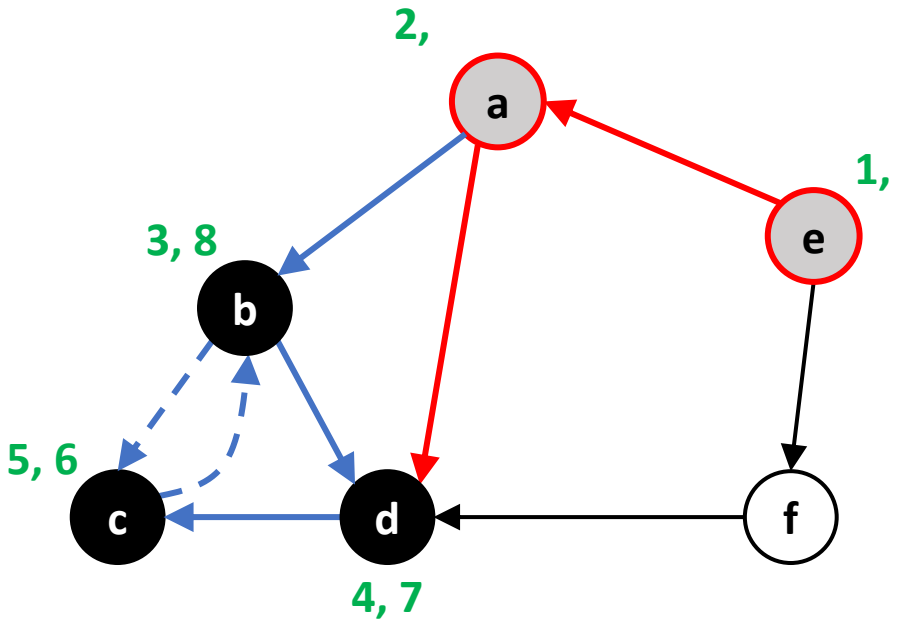


Colors of nodes:

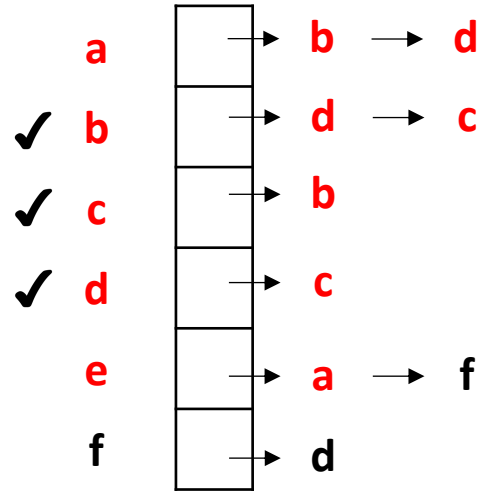
G	G	B	B	B	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

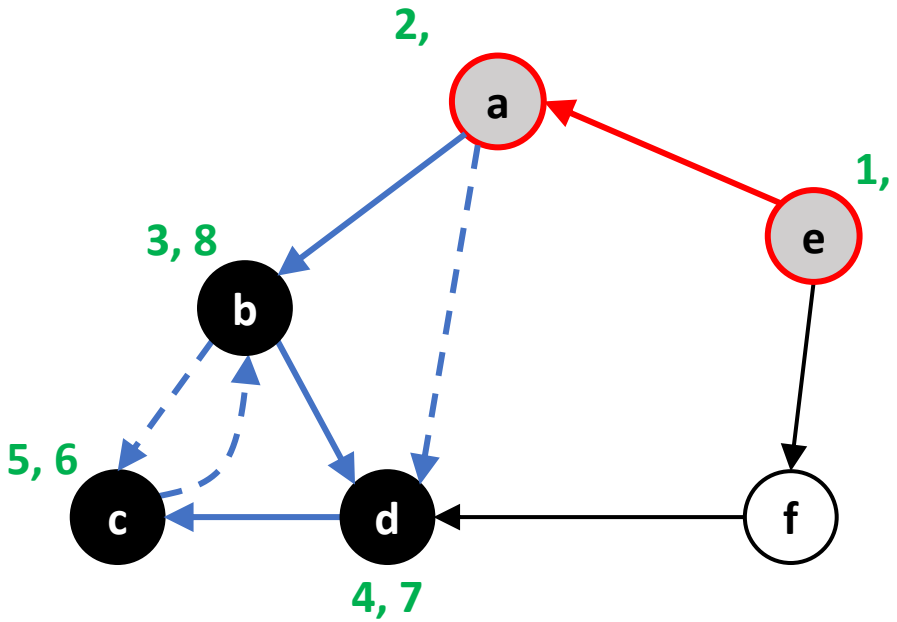


Colors of nodes:

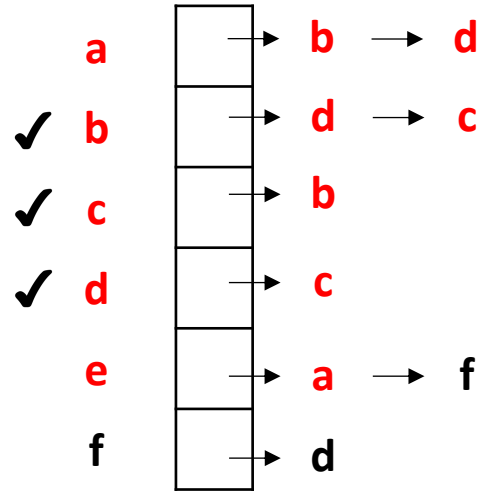
G	G	B	B	B	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

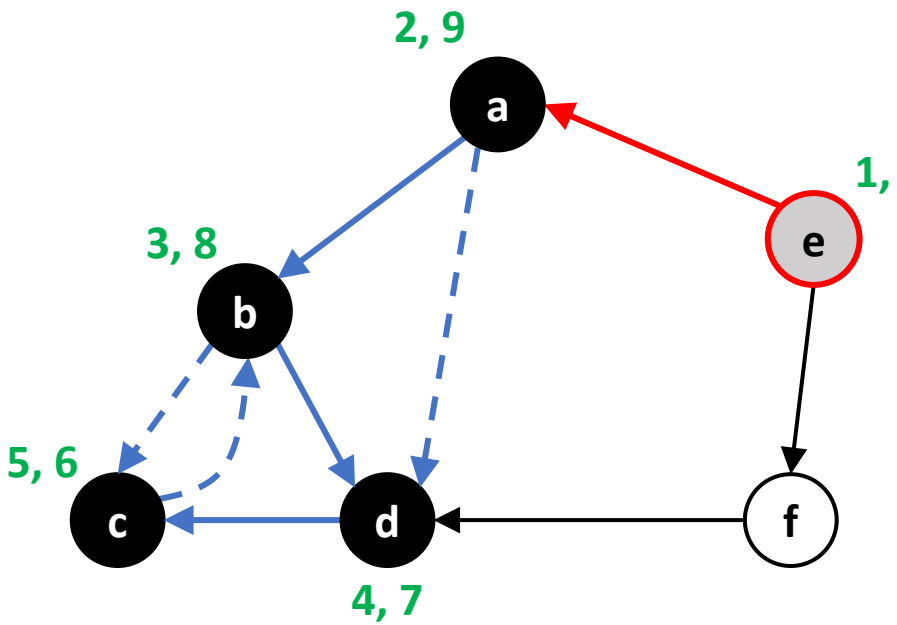


Colors of nodes:

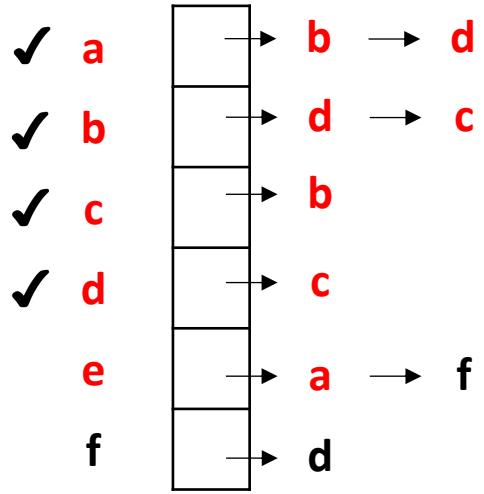
G	G	B	B	B	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

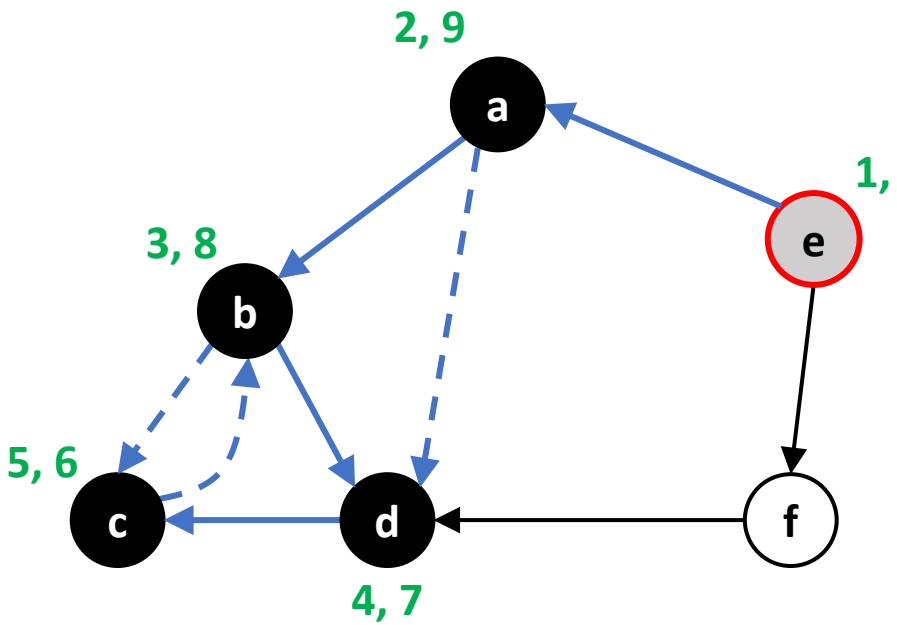


Colors of nodes:

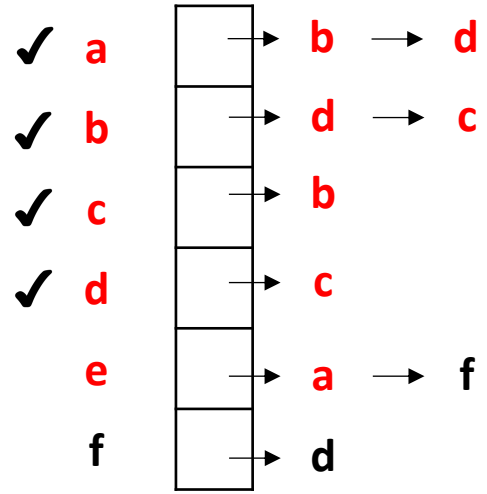
G	B	B	B	B	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

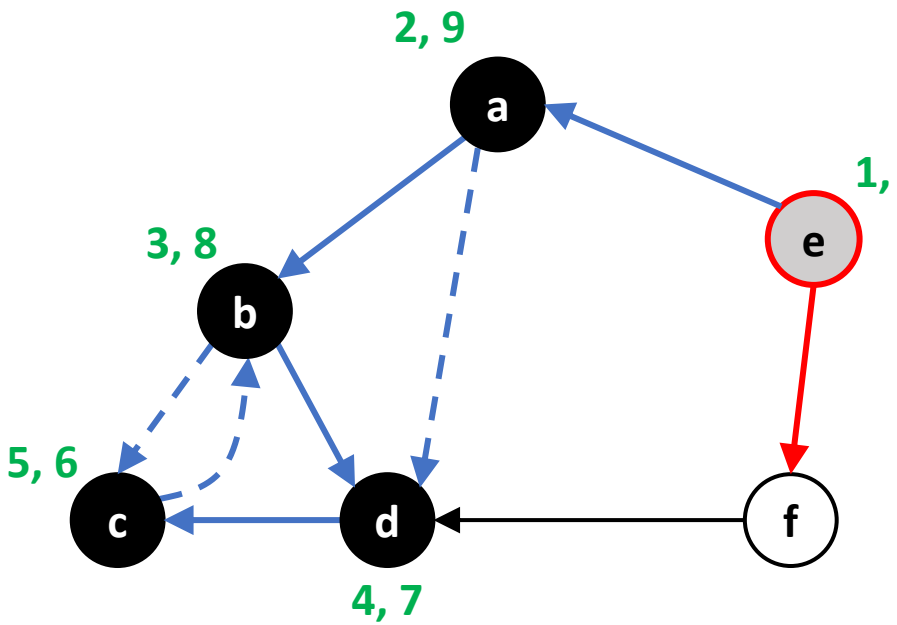


Colors of nodes:

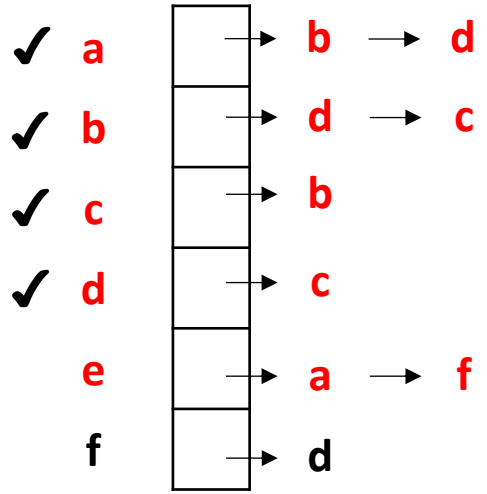
G	B	B	B	B	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

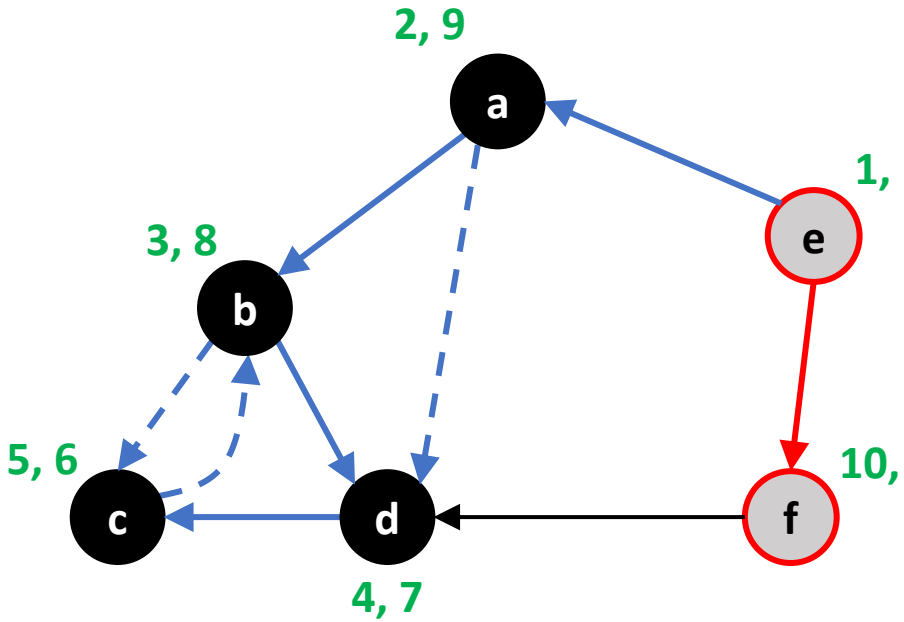


Colors of nodes:

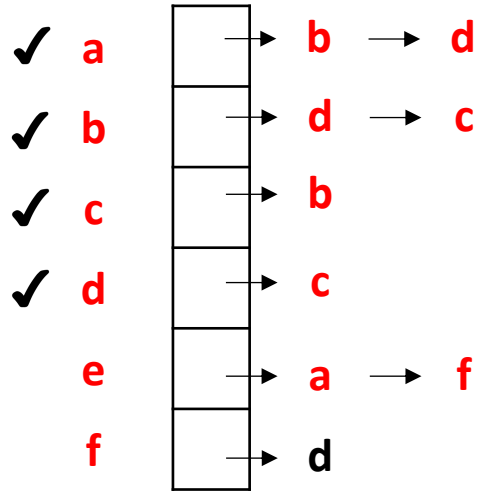
G	B	B	B	B	W
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

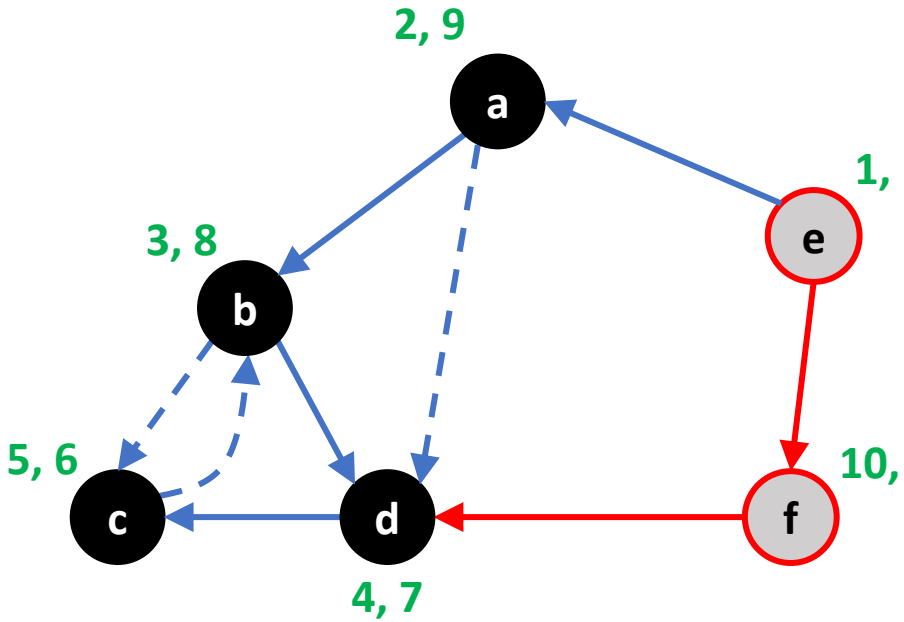


Colors of nodes:

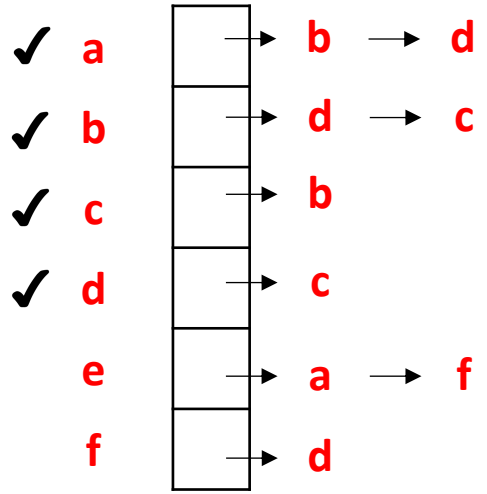
G	B	B	B	B	G
e	a	b	c	d	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

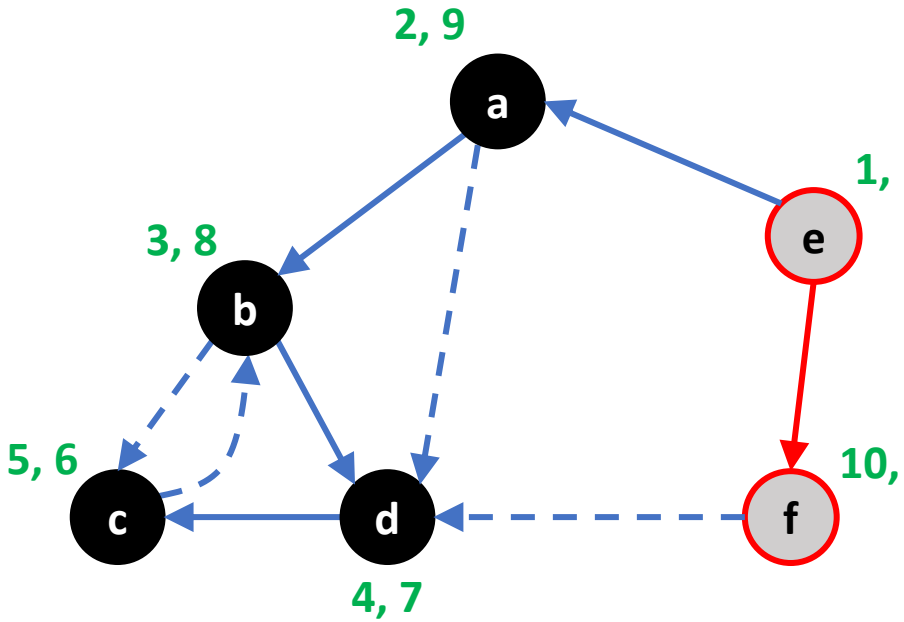


Colors of nodes:

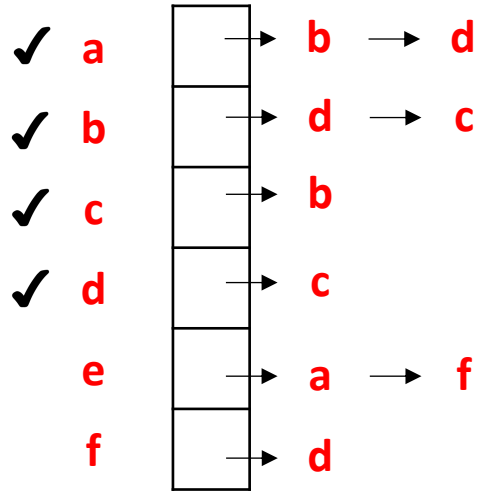
G	B	B	B	B	G
e	a	b	c	d	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

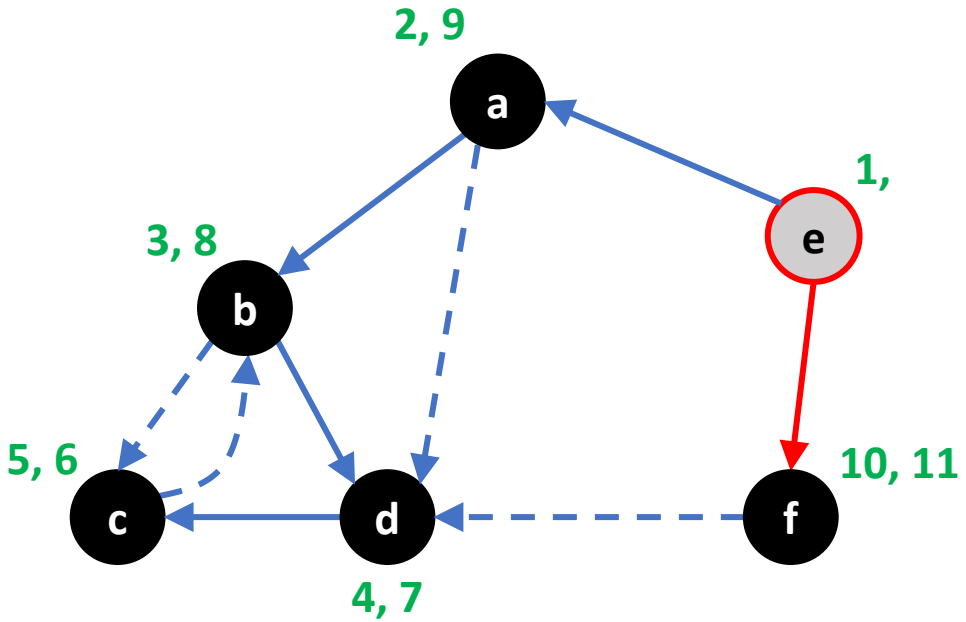


Colors of nodes:

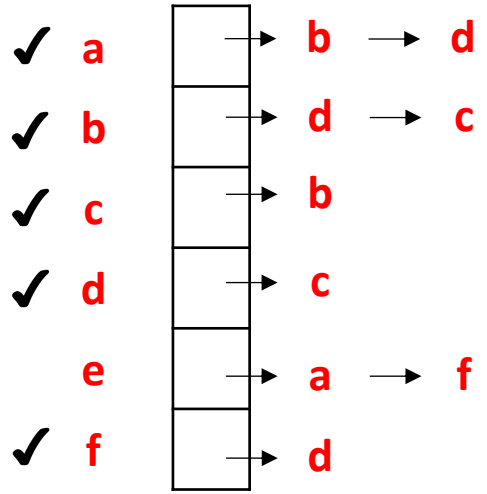
G	B	B	B	B	G
e	a	b	c	d	f

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

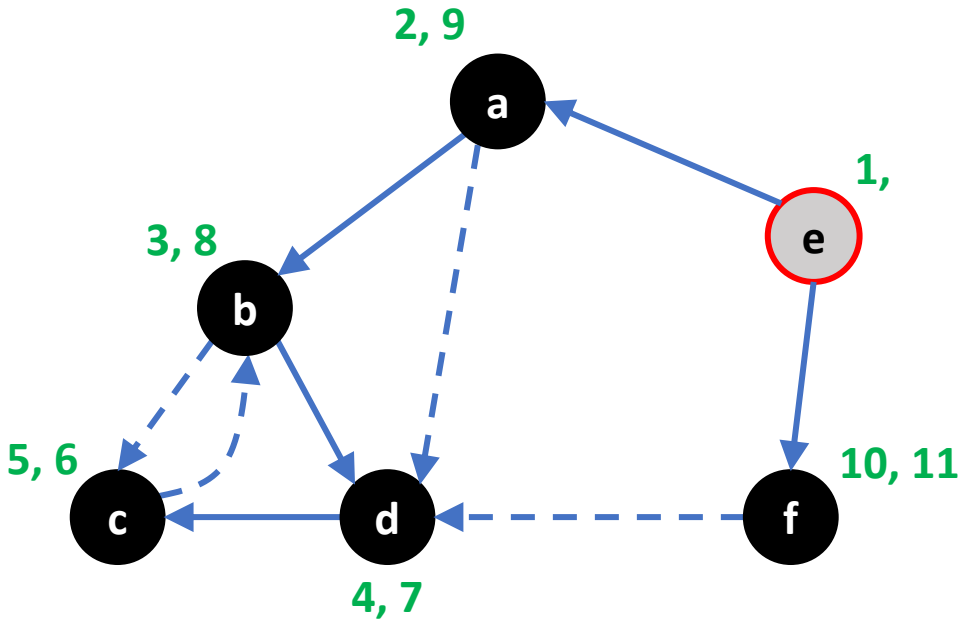


Colors of nodes:

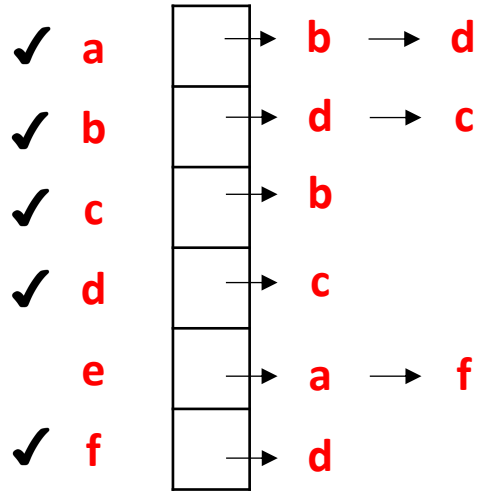
G	B	B	B	B	B
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

DFS(G)



Adj List of G :

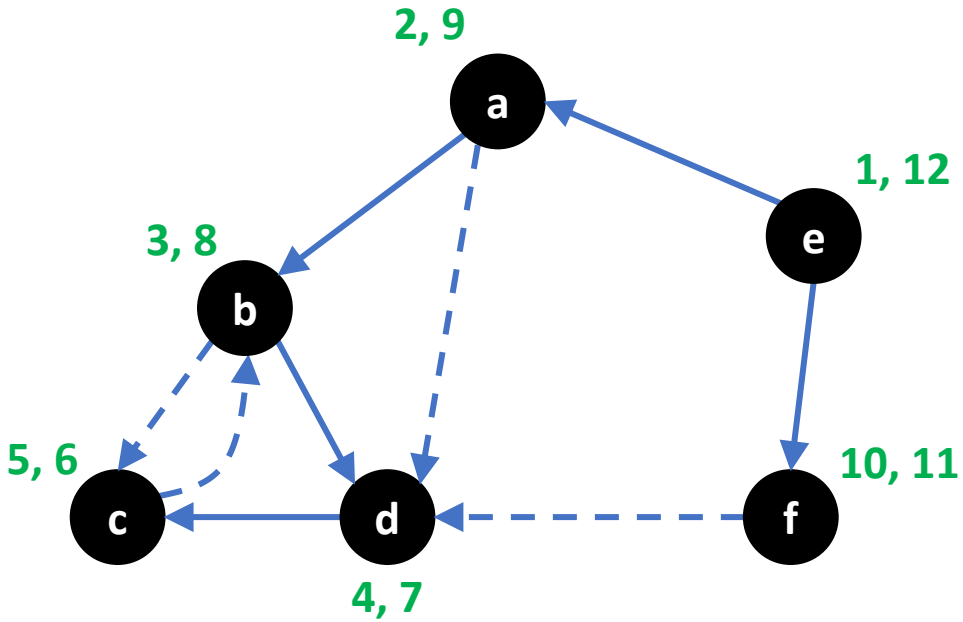


Colors of nodes:

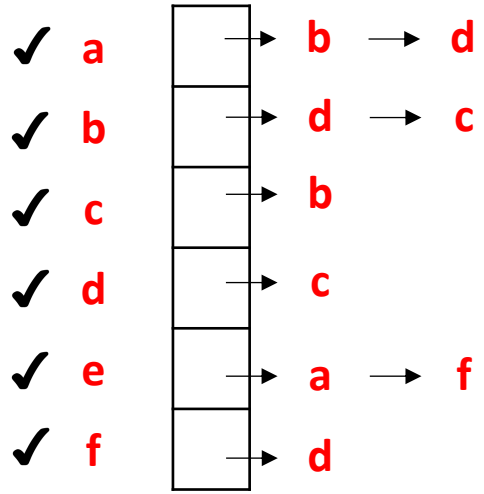
G	B	B	B	B	B
e	**a**	**b**	**c**	**d**	**f**

DFS-Explore(G, **e)**

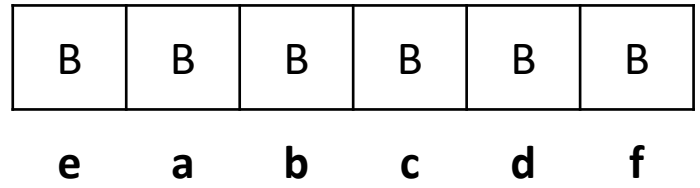
DFS(G)



Adj List of G :

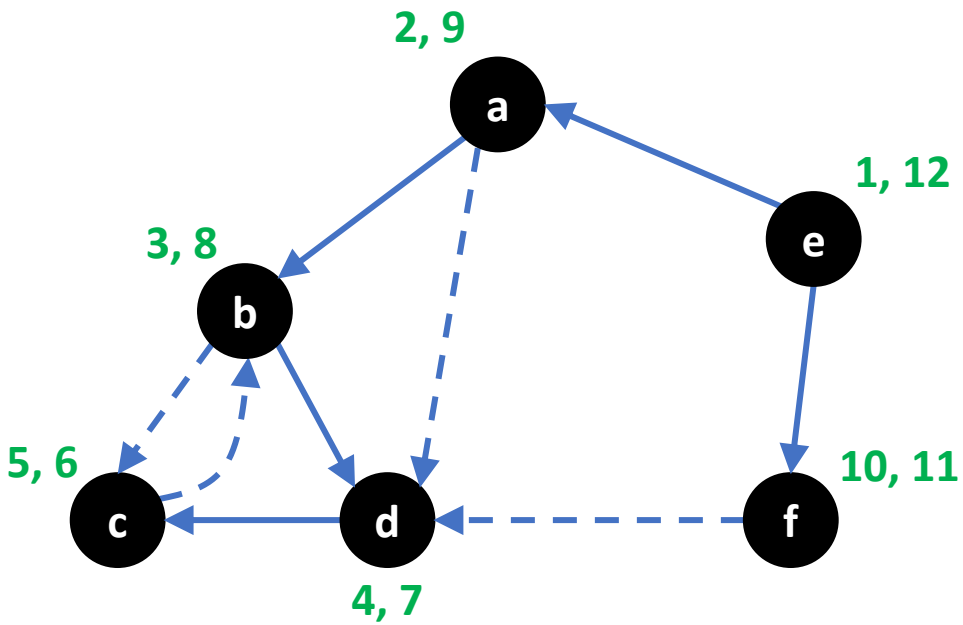


Colors of nodes:

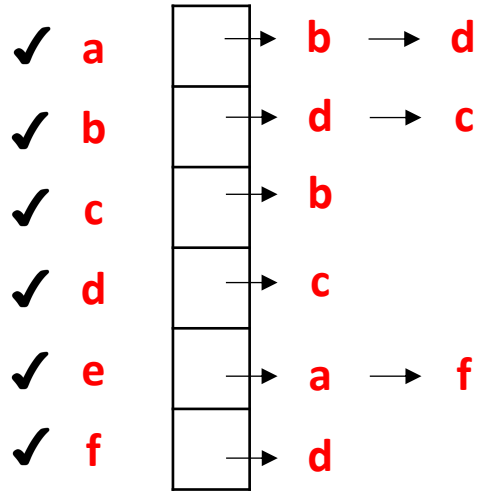


DFS-Explore(G, **e)**

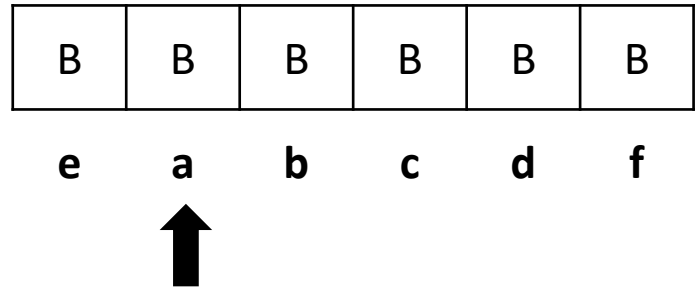
DFS(G)



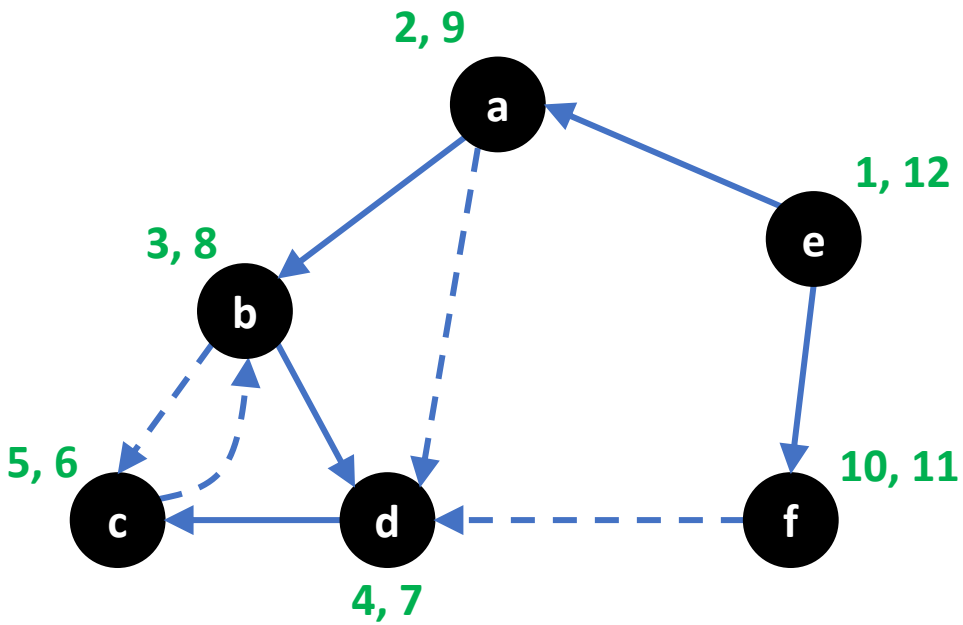
Adj List of G :



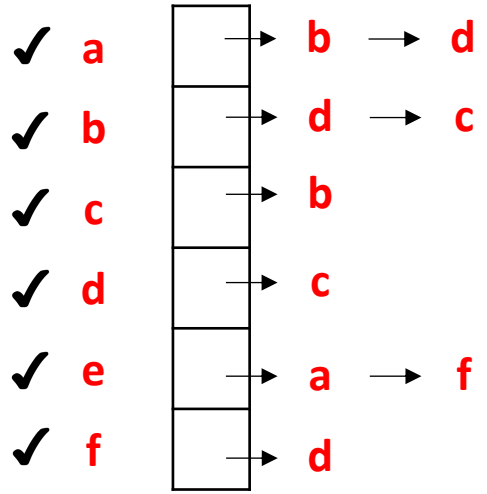
Colors of nodes:



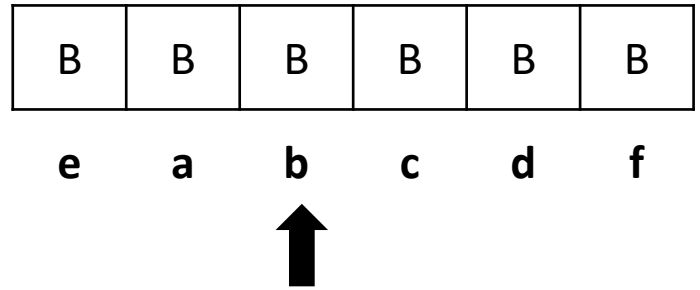
DFS(G)



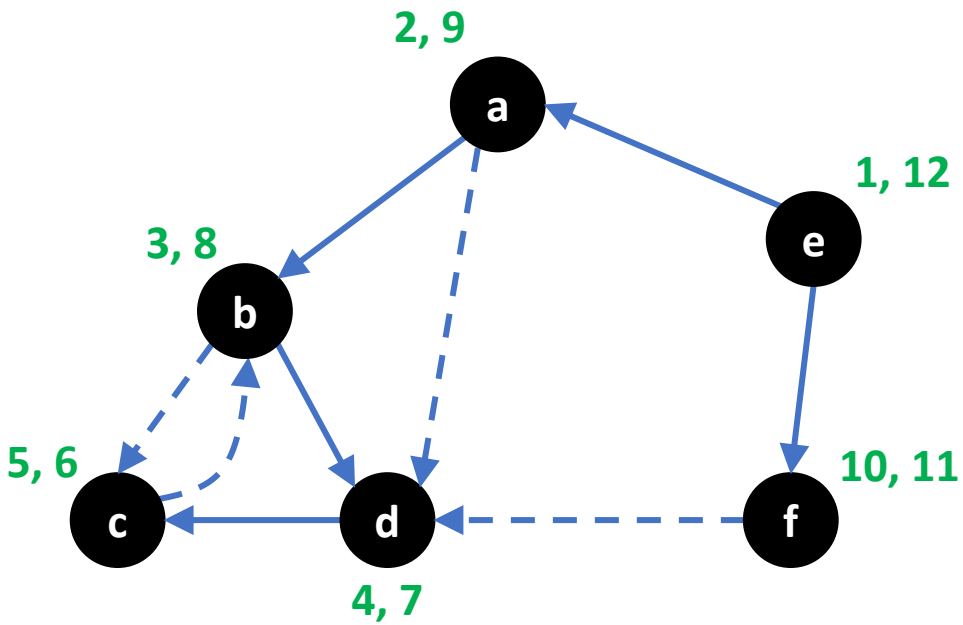
Adj List of G :



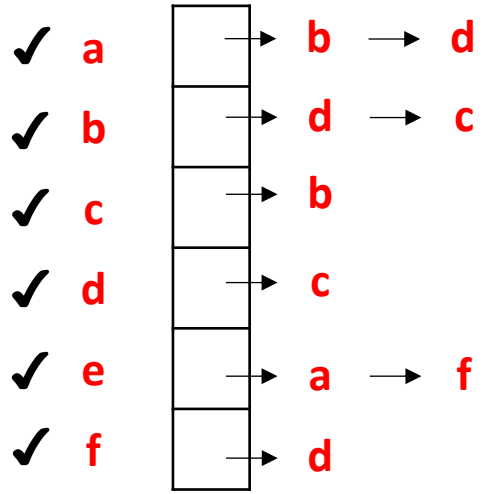
Colors of nodes:



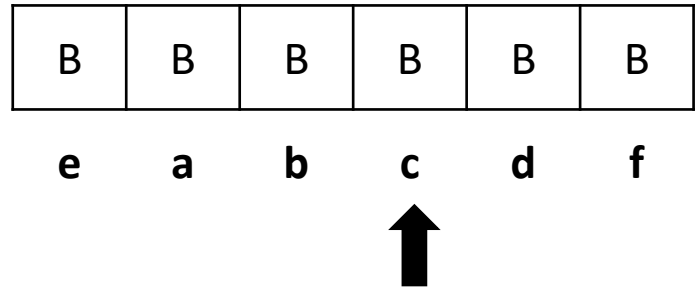
DFS(G)



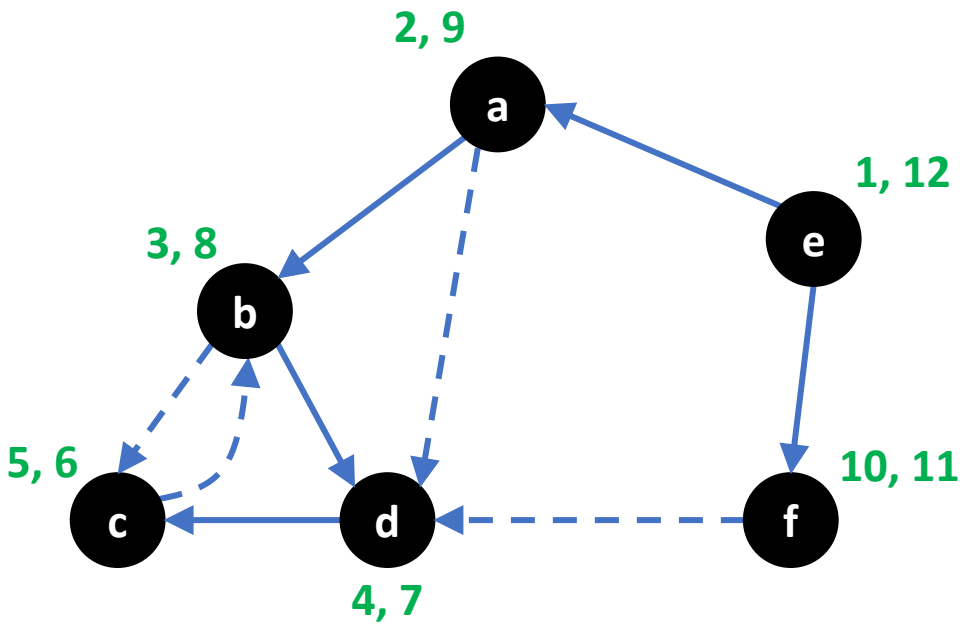
Adj List of G :



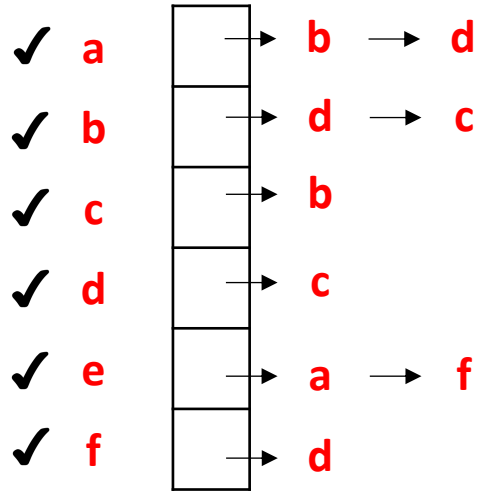
Colors of nodes:



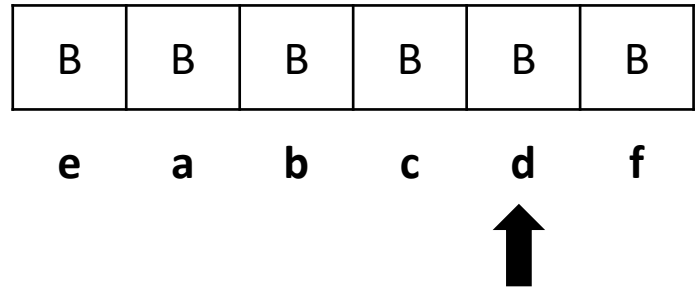
DFS(G)



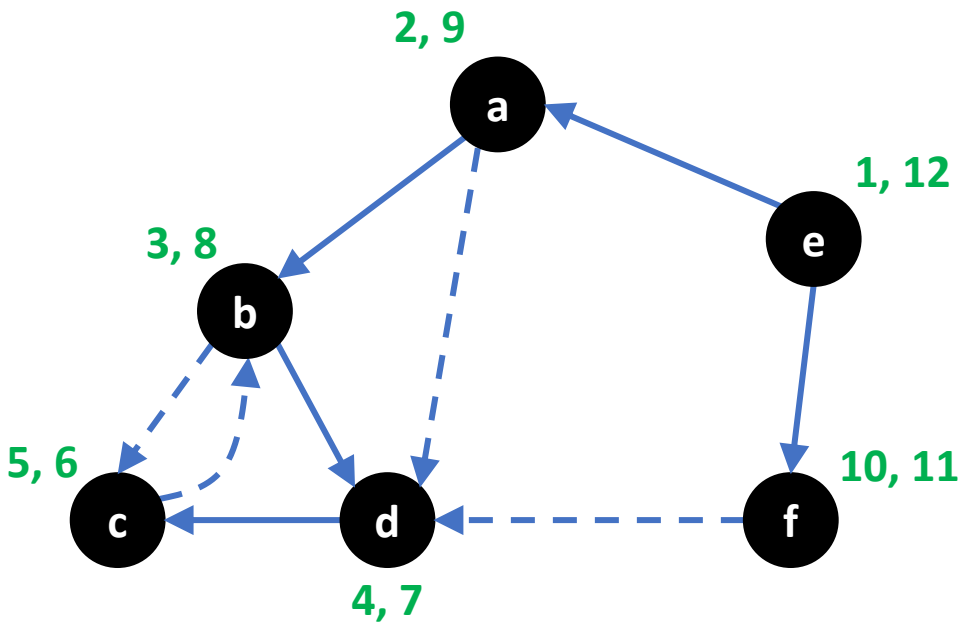
Adj List of G :



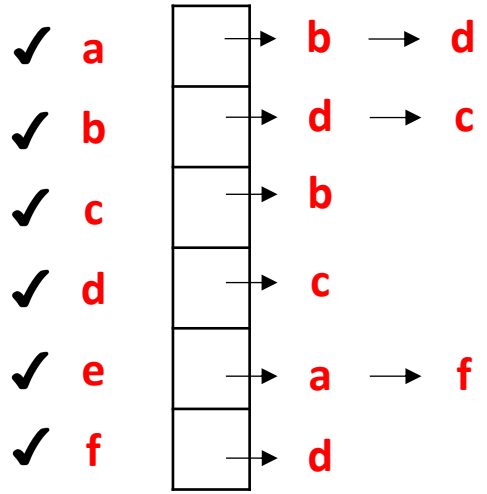
Colors of nodes:



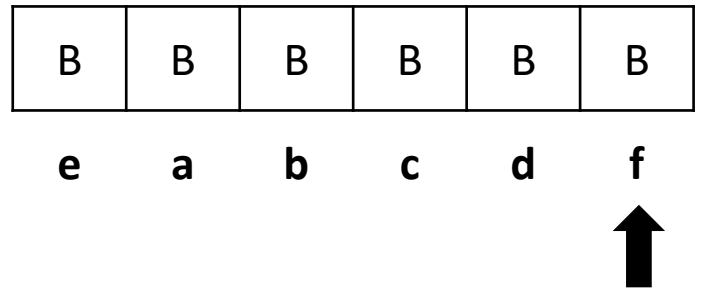
DFS(G)



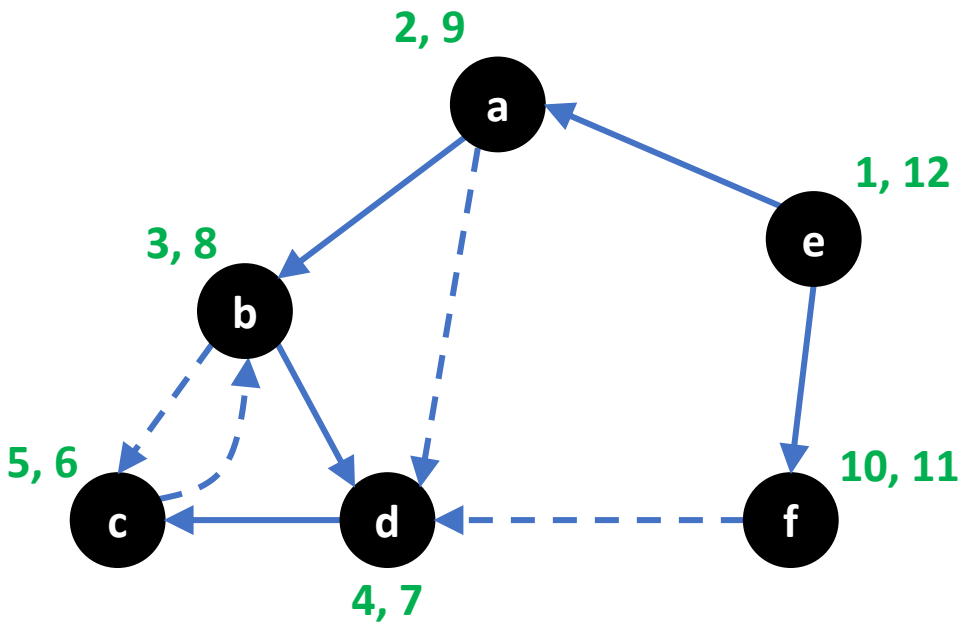
Adj List of G :



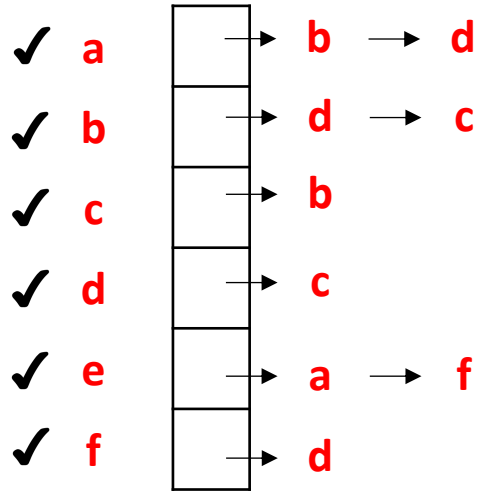
Colors of nodes:



DFS(G)



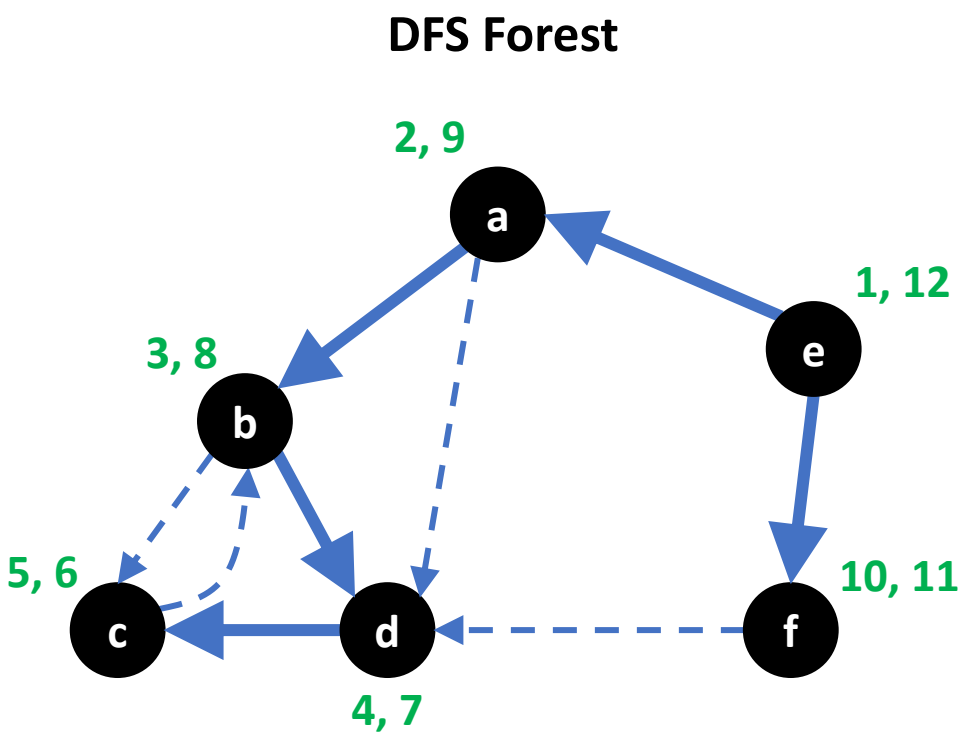
Adj List of G :



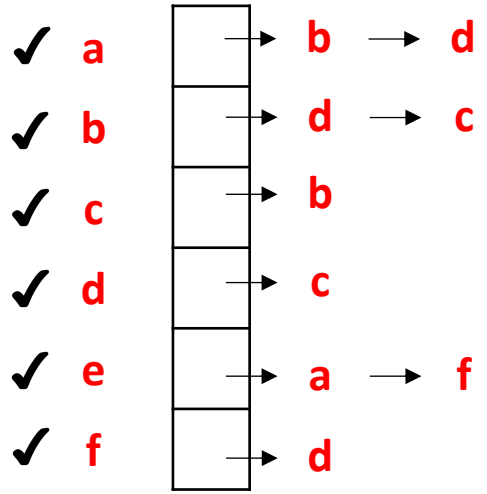
Colors of nodes:

B	B	B	B	B	B
e	**a**	**b**	**c**	**d**	**f**

DFS(G)



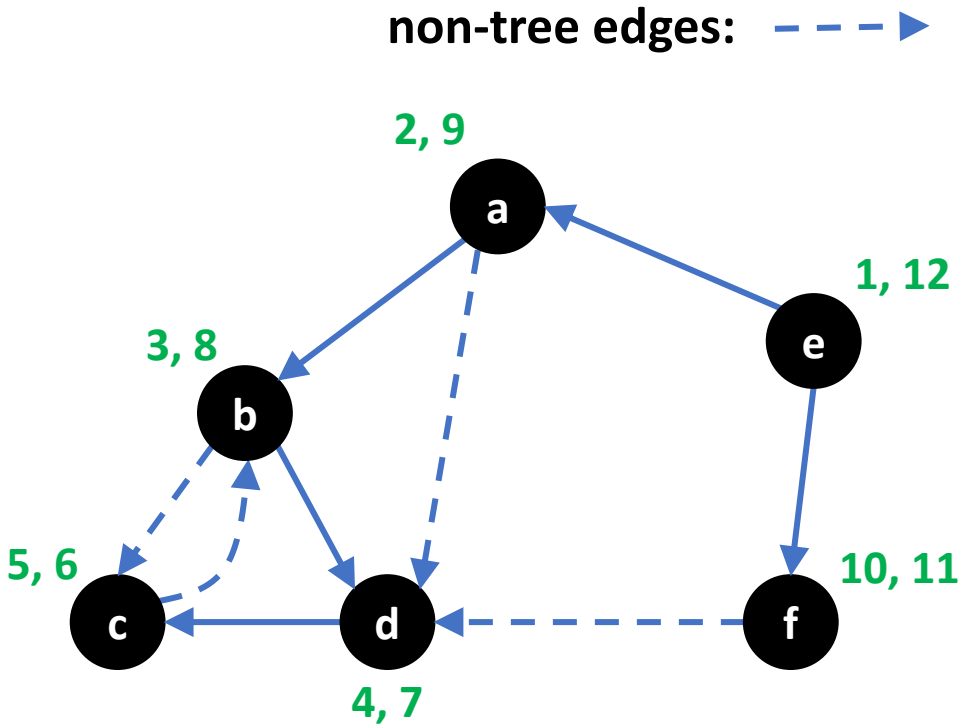
Adj List of G :



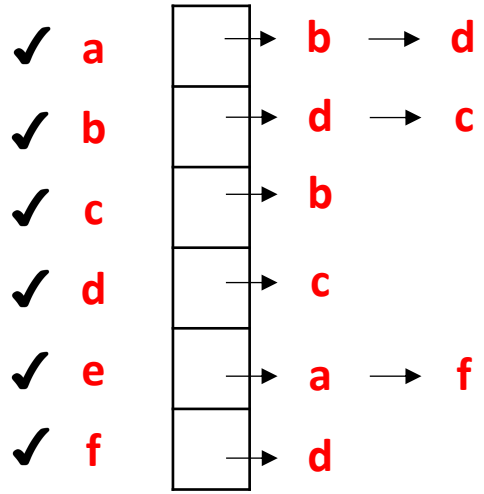
Colors of nodes:

B	B	B	B	B	B
e	**a**	**b**	**c**	**d**	**f**

DFS(G)



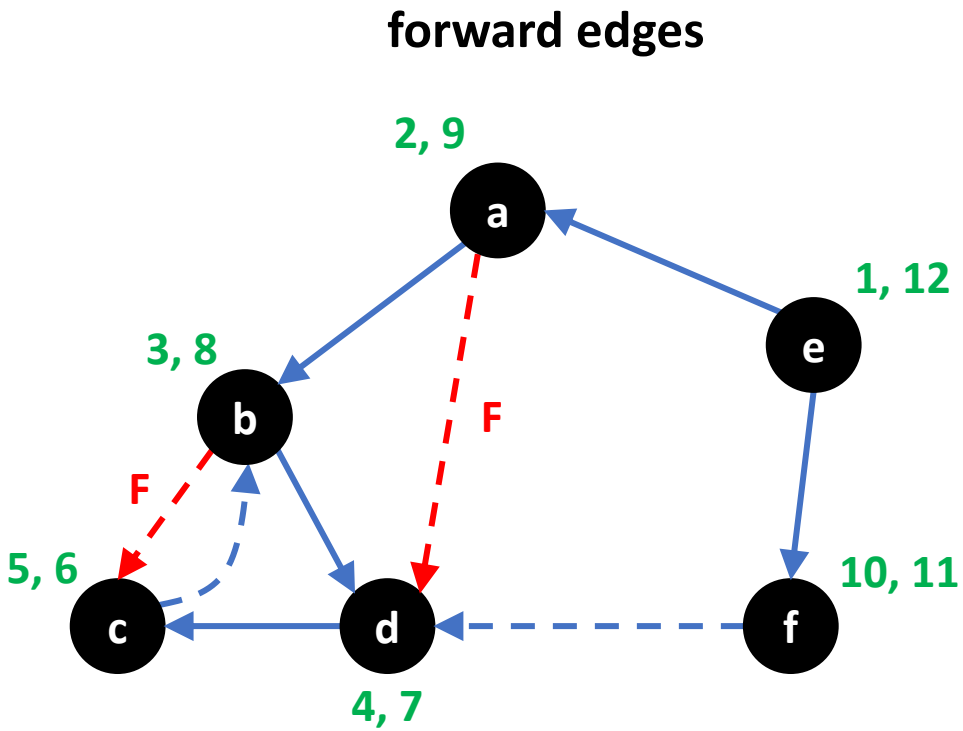
Adj List of G :



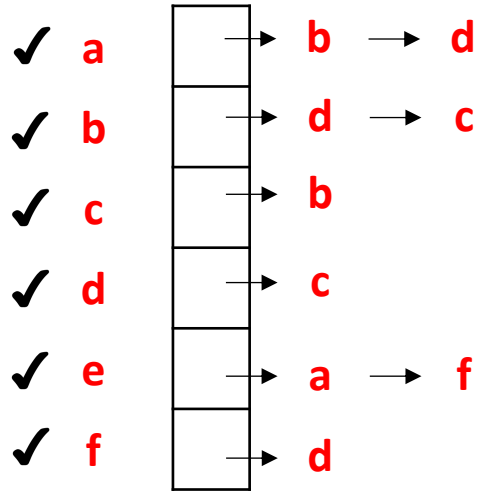
Colors of nodes:

B	B	B	B	B	B
e	**a**	**b**	**c**	**d**	**f**

DFS(G)



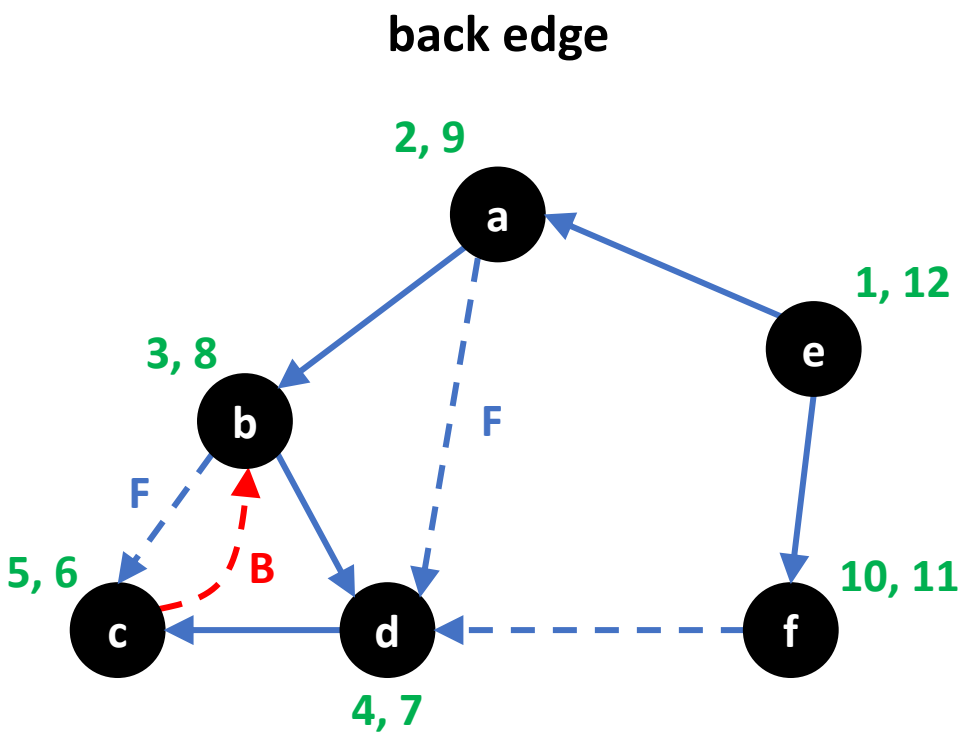
Adj List of G :



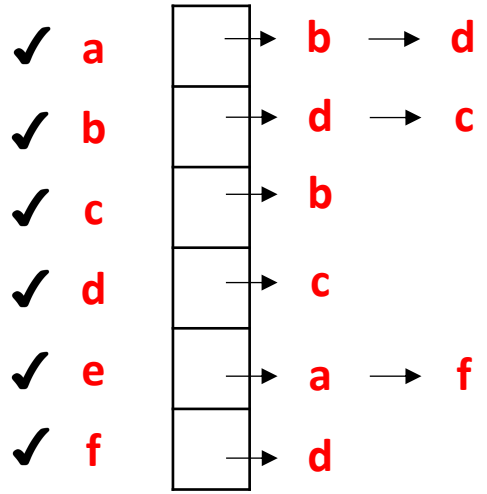
Colors of nodes:

B	B	B	B	B	B
e	a	b	c	d	f

DFS(G)



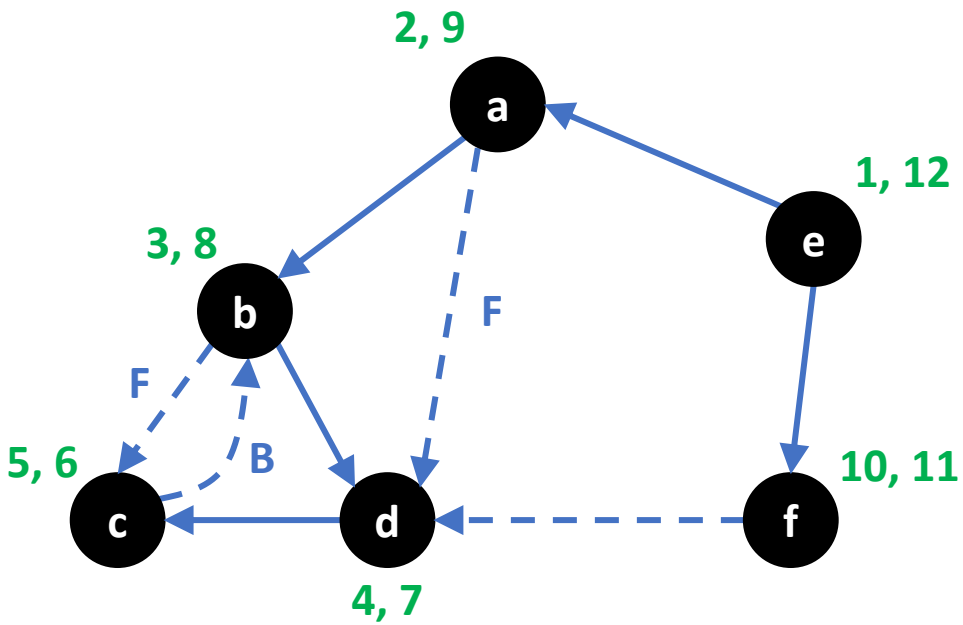
Adj List of G :



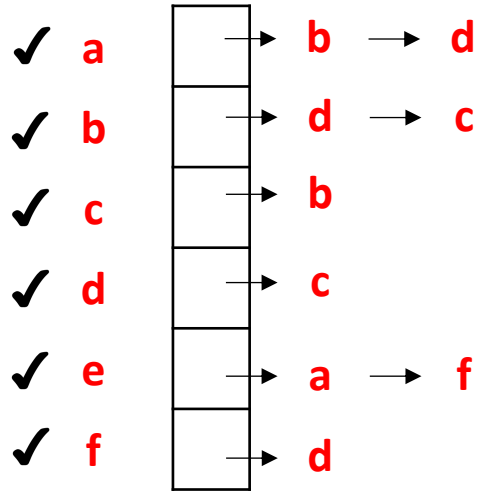
Colors of nodes:

B	B	B	B	B	B
e	**a**	**b**	**c**	**d**	**f**

DFS(G)



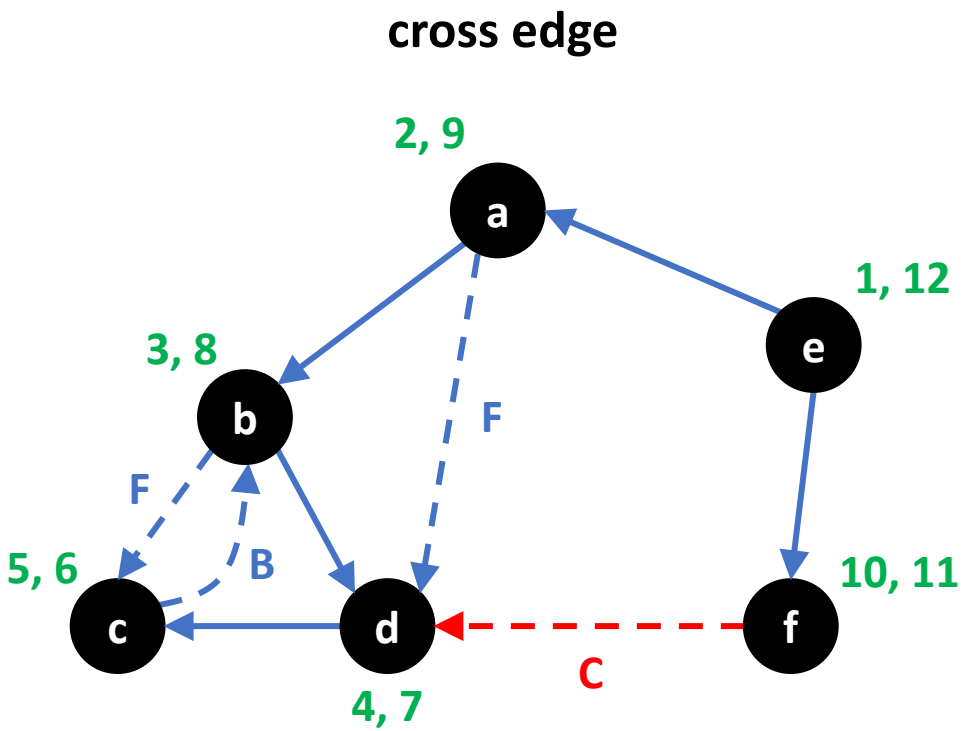
Adj List of G :



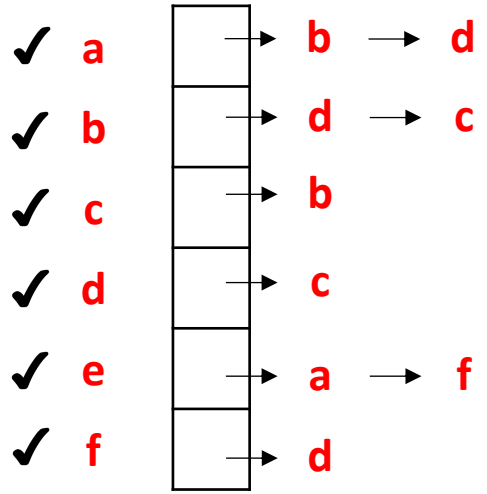
Colors of nodes:

B	B	B	B	B	B
e	**a**	**b**	**c**	**d**	**f**

DFS(G)



Adj List of G :

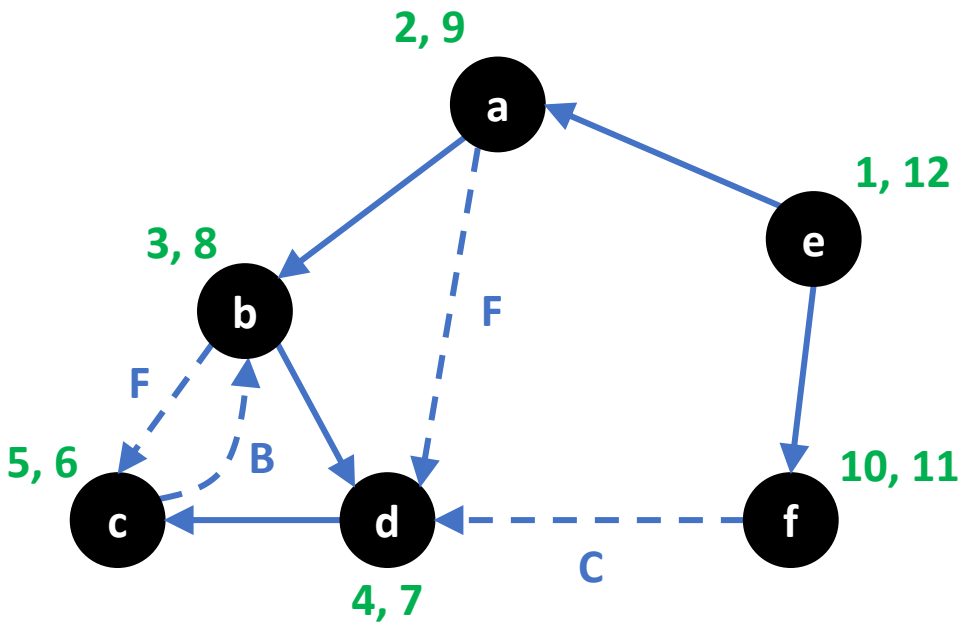


Colors of nodes:

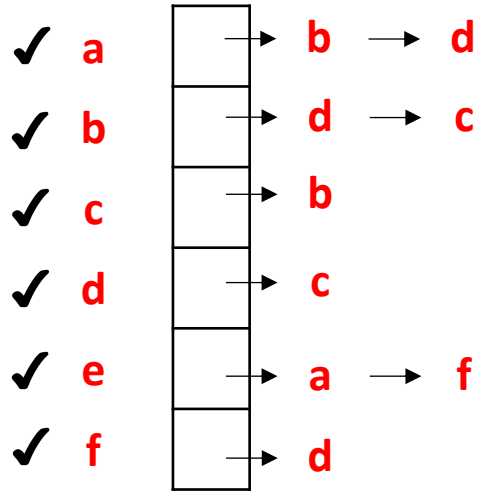
B	B	B	B	B	B
e	**a**	**b**	**c**	**d**	**f**

DFS(G)

DFS Forest and non-tree edges



Adj List of G :



Colors of nodes:

B	B	B	B	B	B
e	**a**	**b**	**c**	**d**	**f**