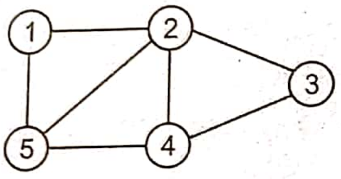
26.2 How is a Graph Represented ?

26.2.1 Representing Graphs as an Adjacency List

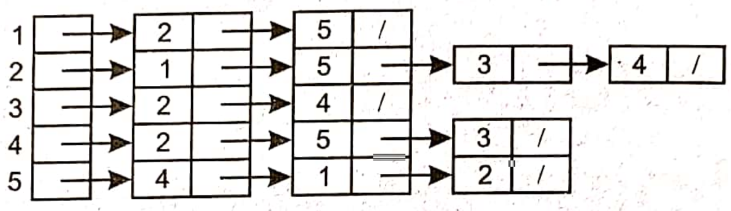
An **adjacency list** is an array, which contains a list for each vertex. The list for a given vertex contains all the other vertices that are connected to the first vertex by a single edge.

26.2.2 Representing Graphs as an Adjacency Matrix

An **adjacency matrix** is a matrix with a row and column for each vertex. The matrix entry is 1 if there is an edge between the row vertex and the column vertex. The diagonal may be zero or one. Each edge is represented twice.



(a)

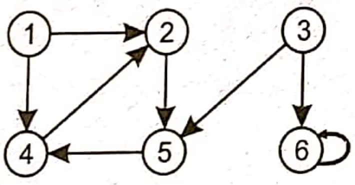


(b)

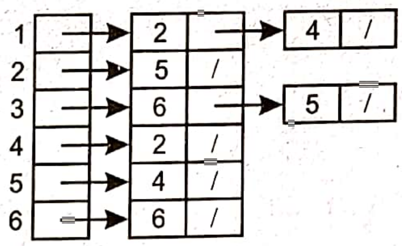
$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \end{array}$$

(c)

Undirected graph : Matrix is symmetric



(a)



(b)

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 & 1 & 1 \\ 4 & 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

(c)

Directed graph : Matrix is asymmetric

}

For many applications, one must systematically search through a graph to determine the structure of the graph.

Two common elementary algorithms for tree searching are :

- Breadth-first search (BFS), and
- Depth-first search (DFS).

Both of these algorithms work on directed or undirected graphs. Many advanced graph algorithms are based on the ideas of BFS or DFS. Each of these algorithms traverses edges in the graph, discovering new vertices as it proceeds. The difference is in the order in which each algorithm discovers the edge

26.3 Breadth First Search

Breadth first search trees have a nice property: Every edge of *G* can be classified into one of three groups. Some edges are in *T* themselves. Some connect two vertices at the same level of *T*. And the remaining ones connect two vertices on two adjacent levels. It is not possible for an edge to skip a level.

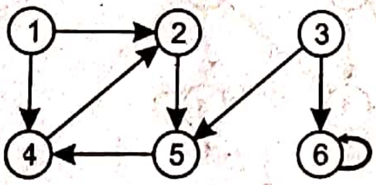
Therefore, the breadth first search tree really is a shortest path tree starting from its root. Every vertex has a path to the root, with path length equal to its level (just follow the tree itself), and no path can skip a level so this really is a shortest path

Analysis

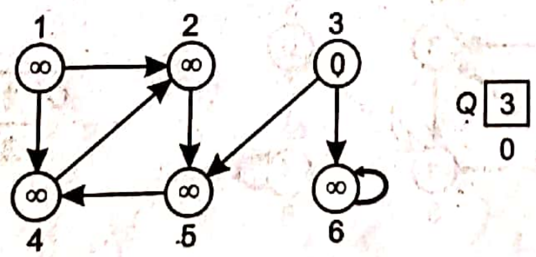
We use aggregate analysis. After initialization, no vertex is ever whitened, and thus the test in line 13 ensures that each vertex is enqueued at most once, and hence dequeued at most once. The operations of enqueuing and dequeuing take $O(1)$ time, so the total time devoted to queue operations is $O(V)$. Because the adjacency list of each vertex is scanned only when the vertex is dequeued, each adjacency list is scanned at most once. Since the sum of the lengths of all the adjacency list is $Q(E)$, the total time spent in scanning adjacency list is $O(E)$. The overhead for initialization is $O(V)$, and thus total running time of BFS is $O(V + E)$. Thus, BFS runs in linear time in the size of the adjacency-list representation of G.

```
BFS (G, s)
1.  for each vertex $u \in V[G] - \{s\}$
2.      do $color[u] \leftarrow$ WHITE
3.         $d[u] \leftarrow \infty$
4.         $\pi[u] \leftarrow$ NIL
5.  $color[s] \leftarrow$ GRAY
6.  $d[s] \leftarrow 0$
7.  $\pi[s] \leftarrow$ NIL
8.  $Q \leftarrow \phi$
9.  ENQUEUE(Q, s)
10. while $Q \neq \phi$
11.     do $u \leftarrow$ DEQUEUE(Q)
12.        for each $v \in Adj[u]$
13.            do if $color[v]$ = WHITE
14.                then $color[v] \leftarrow$ GRAY
15.                     $d[v] \leftarrow d[u] + 1$
16.                     $\pi[v] \leftarrow u$
17.                     ENQUEUE(Q, v)
18.        $color[u] \leftarrow$ BLACK
```

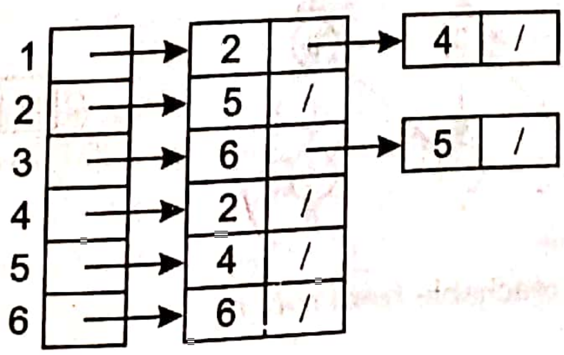
Example. *Consider the graph G in Fig. Describe the whole process of breadth first search. Using vertex 3 as the source.*



Solution.



First, we create adjacency-list representation



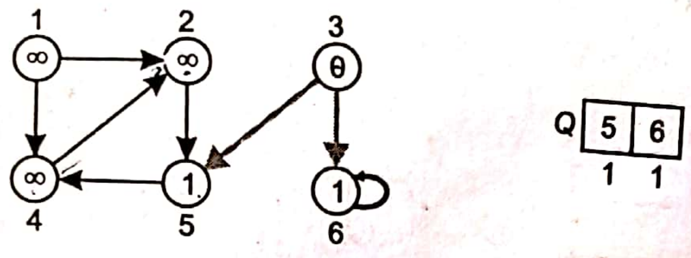
So, $adj[4] = \{6, 5\}$ so color [6] = GRAY

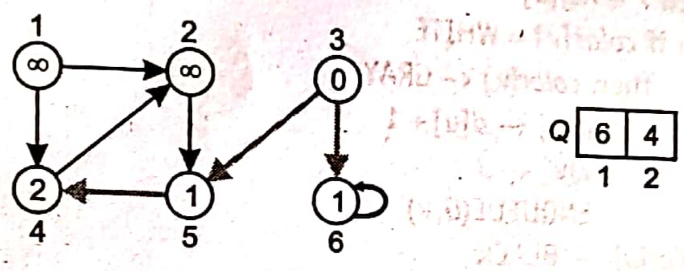
color [5] = GRAY

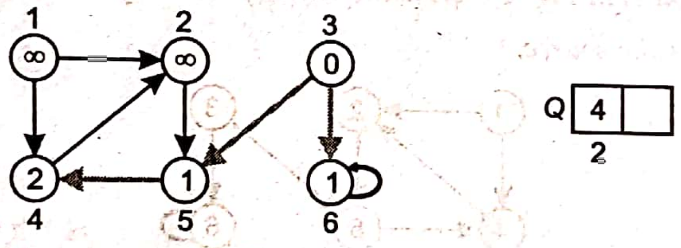
and $d[6] = 1$ $d[5] = 1$

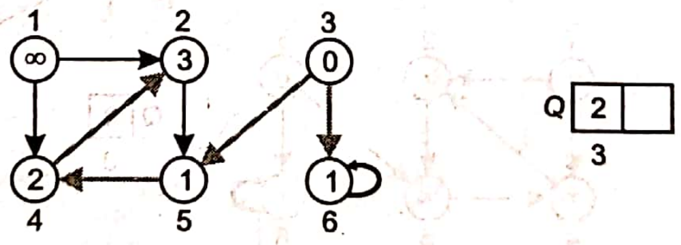
$\pi[6] \leftarrow 3$ $\pi[5] \leftarrow 3$

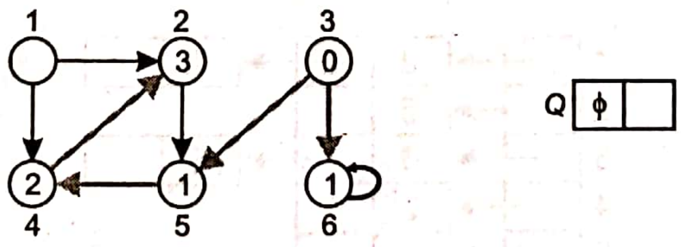
Now





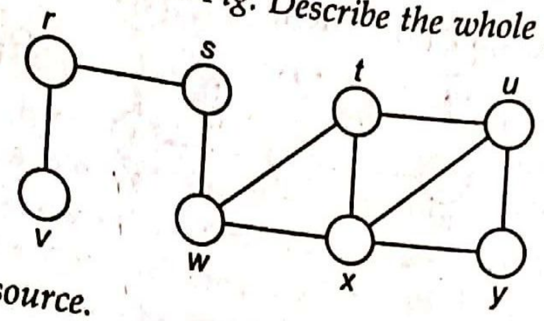






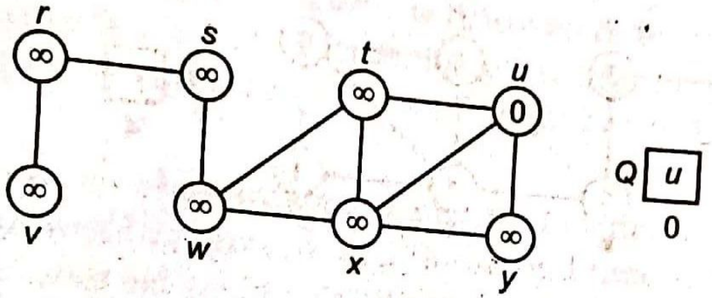
Thus vertex 1 cannot be reachable from source.

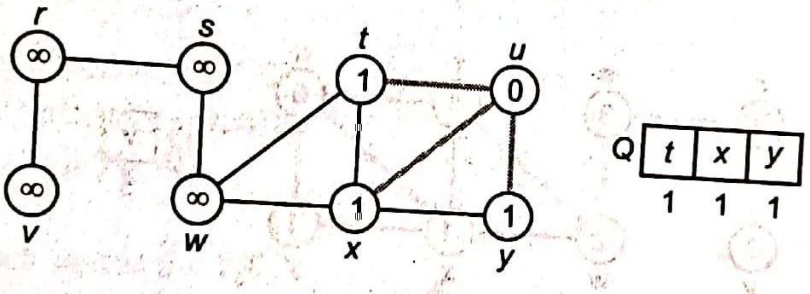
G shown in the Fig. Describe the whole process of BFS.

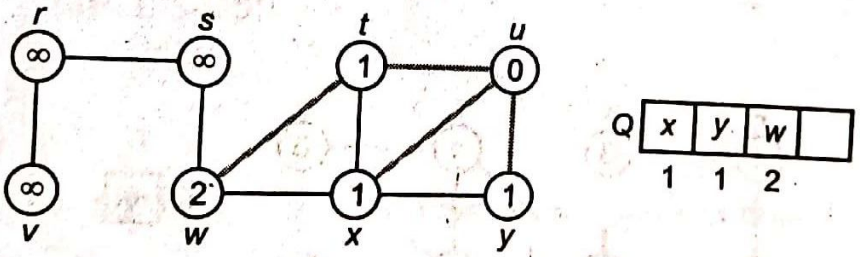


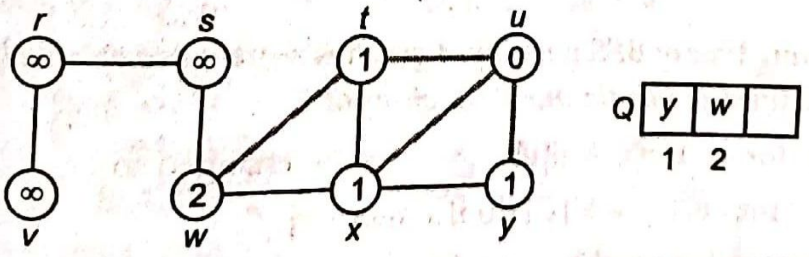
Using vertex u as the source.

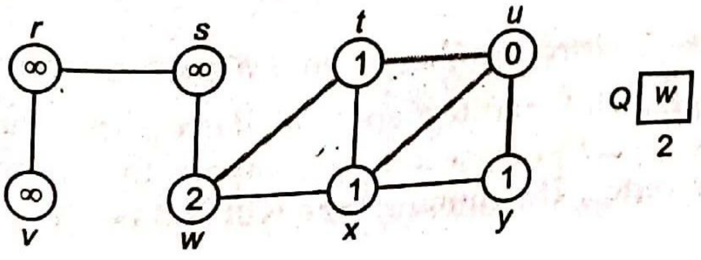
Solution.

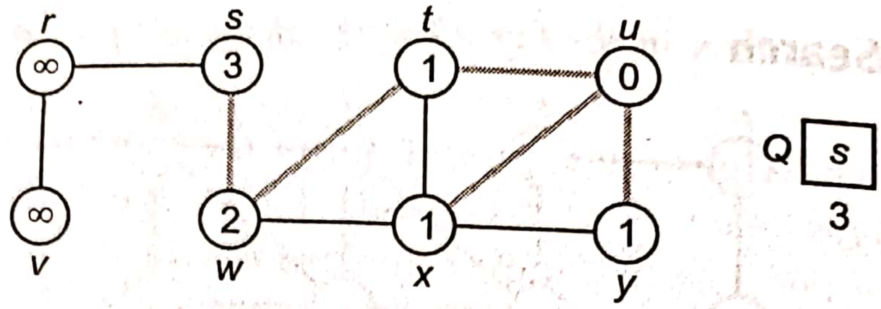


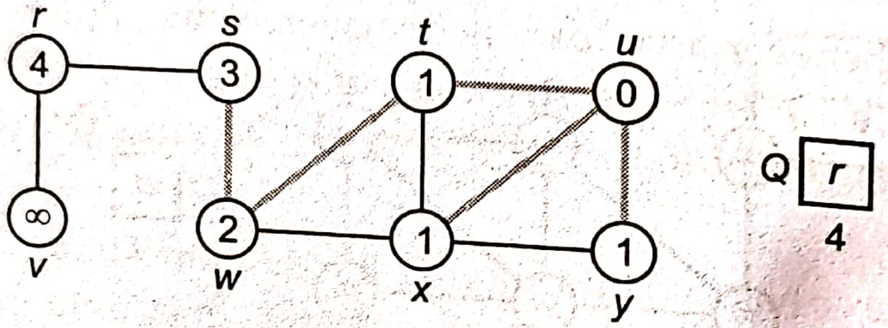


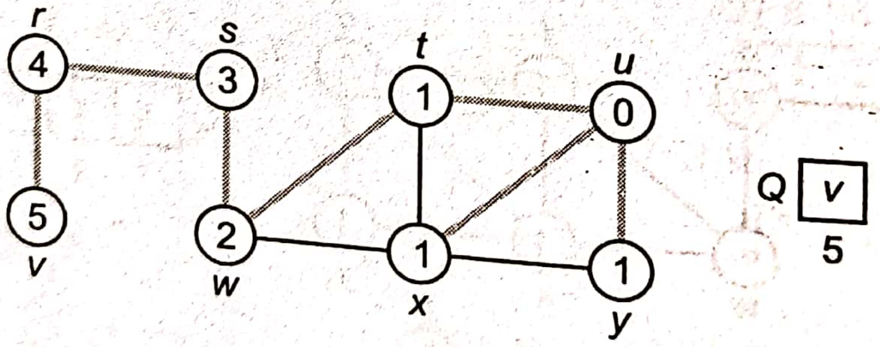


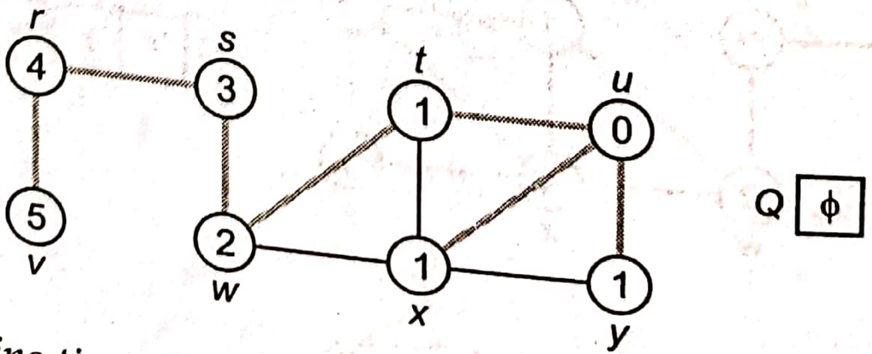












nple. *What is the running time of BFS if its input graph is represented by an adjacency mat*
algorithm is modified to handle thi

Solution. Th

26.4 Depth First Search

In depth-first search, edges are explored out of the most recently discovered vertex v that still has unexplored edges leaving it. When all of v's edges have been explored, the search backtracks to explore edges leaving the vertex from which v was discovered. This process continues until we have discovered all the vertices that are reachable from the original source vertex. If any undiscovered vertices remain, then one of them is selected as a new source and the search is repeated from that source. This entire process is repeated until all vertices are discovered.

As in the BFS, whenever a vertex v is discovered during a scan of the adjacency list of an already discovered vertex u, DFS records this by setting v's predecessor field $\pi[v]$ to u. In DFS predecessor subgraph is composed of several trees, because the search may be repeated from multiple sources.

In DFS, each vertex v has two timestamps: the first timestamp $d[v]$ *i.e.* discovery time records when v is first discovered *i.e.* grayed, and the second timestamp $f[v]$ *i.e.* finishing time records when the search finishes examining v's adjacency list *i.e.* blacked. For every vertex $d[u] < f[u]$.

```
DFS(G)

1.   for each vertex $u \in V[G]$
2.        do $color[u] \leftarrow$ WHITE
3.           $\pi[u] \leftarrow$ NIL
4.   $time \leftarrow 0$
5.   for each vertex $u \in V[G]$
6.        do if $color[u] =$ WHITE
7.              then DFS-VISIT($u$)
```

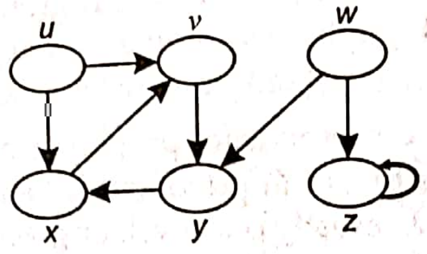
```
DFS-VISIT(u)

1.   $color[u] \leftarrow$ GRAY                ▷ White vertex $u$ has just been discovered.
2.   $time \leftarrow time + 1$
3.   $d[u] \leftarrow time$
4.   for each $v \in Adj[u]$              ▷ Explore edge $(u,v)$
5.        do if $color[v] =$ WHITE
6.              then $\pi[v] \leftarrow u$
7.                   DFS-VISIT ($v$)
8.   $color[u] \leftarrow$ BLACK               ▷ Blacken $u$, it is finished.
9.   $f[u] \leftarrow time \leftarrow time + 1$
```

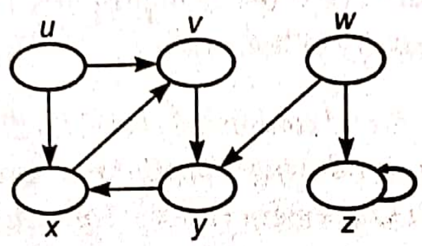
Analysis

In this we use aggregate analysis. The loops on lines 1-3 and lines 5-7 of DFS take time $\Theta(V)$, exclusive the calls to DFS-VISIT. The DFS-Visit calls exactly once for each vertex $v \in V$. During an execution of DFS-VISIT, the loop on lines 4-7 is executed $|Adj[v]|$ times. The total cost of executing lines 4-7 of DFS-VISIT is $\Theta(E)$. The running time of DFS is therefore $\Theta(V + E)$.

Example. *Show the progress of the depth-first-search (DFS) algorithm on a directed graph.*



Solution.



For each vertex $u \in V[G]$

do color $[u]$ = white

$\pi[u] = \text{NIL}$

i.e., assign color white to all vertices in the graph.

For each vertex $u \in v[G]$

if color$[u]$ = white [which is true]

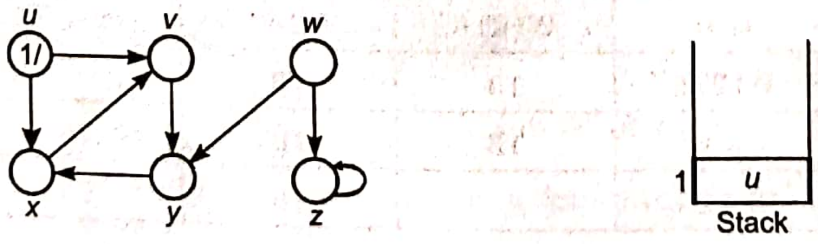
then DFS-VISIT$[u]$

Now apply DFS-VISIT$[u]$

color$[u]$ = gray

time $= 0 + 1 = 1$ and $d[u] = 1$

i.e.,



For each $v \in \text{adj}[u]$

Here $\text{adj}[u] = \{v, x\}$

if color$[v]$ = white [true]

then $\pi[v] = u$

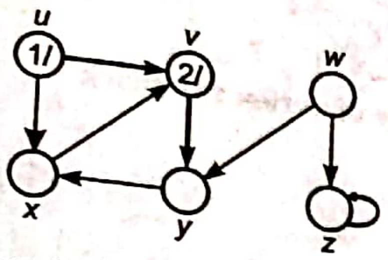
i.e., make u as parent of v.

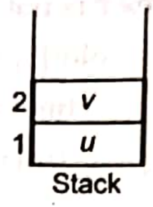
Now apply DFS-VISIT (v)

Make color of vertex v to gray

$$\text{time} = 1+1 = 2$$

$$d[v] = 2$$





Now find adjacent of vertex v

$$\text{adj}[v] = y$$

Check color of y if color$[y]$ = white (true)

Make vertex v as a parent of vertex y

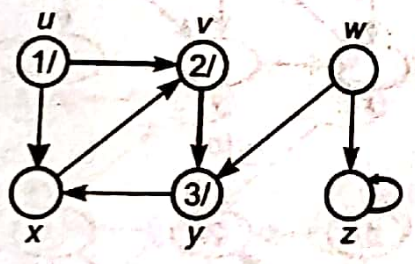
$$\pi(v) = y$$

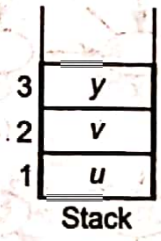
Again apply DFS-VISIT for vertex y

$$\text{color}[y] = \text{gray}$$

$$\text{time} = 2+1 = 3$$

$$d[y] = 3$$





Now find adjacent of y

$$\text{adj}[y] = x$$

check color of x color$[x]$ = white[true]

Make vertex y as parent of vertex x.

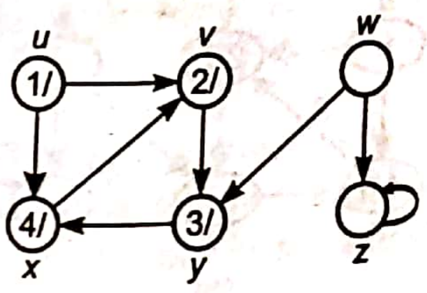
Now apply DFS-VISIT for vertex x

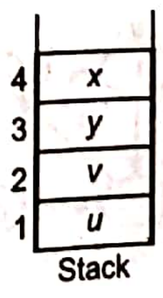
DFS-VISIT(x)

$$\text{color } [x] = \text{gray}$$

$$\text{time} = 3+1 = 4$$

$$d[x] = 4$$





Now find adjacent of vertex x

$$\text{adj}[x] = v$$

Check the color of v,

$$\text{color } [v] = \text{white (false)}$$

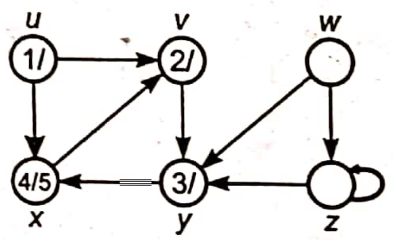
because color of vertex v is not white.

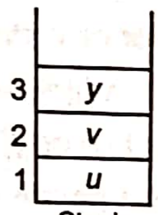
So, $\text{color}[x] = \text{Black}$, *i.e.*, set color of vertex x to black

$$\text{time} = 4 + 1 = 5$$

$\therefore$ $$f[x] = 5$$

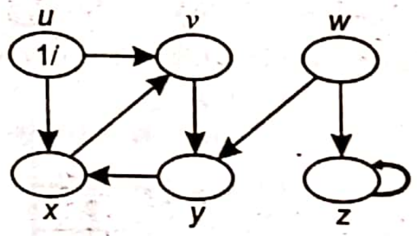
i.e.,



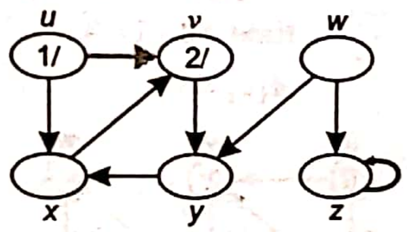


Stack

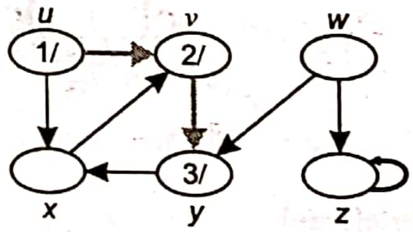
Similarly, we apply above steps to explore all the vertices present in the graph. All the steps in DFS are as follows :



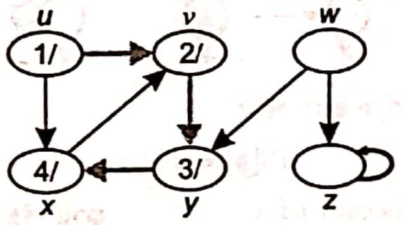
(a)



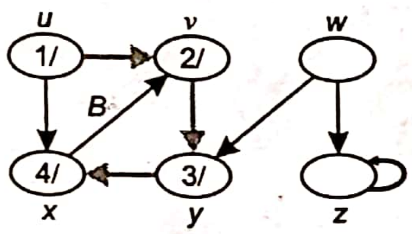
(b)



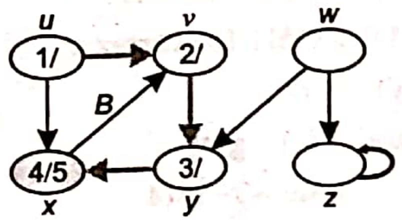
(c)



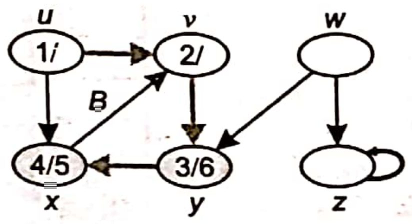
(d)



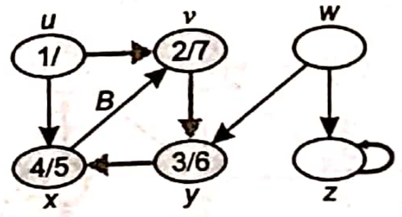
(e)



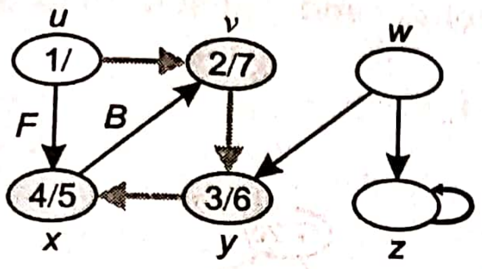
(f)

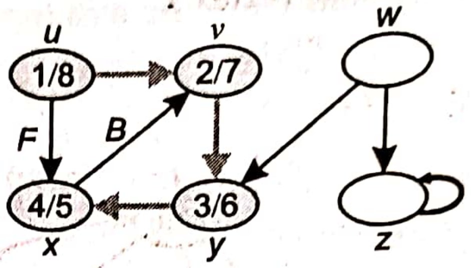


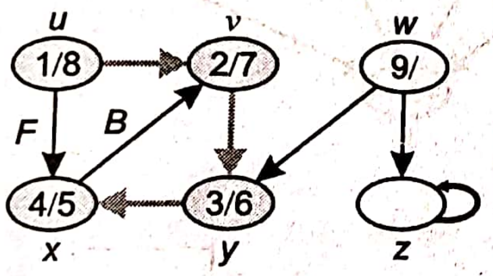
(g)

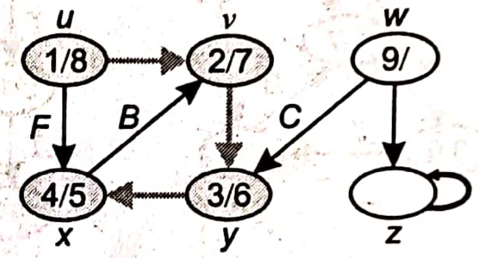


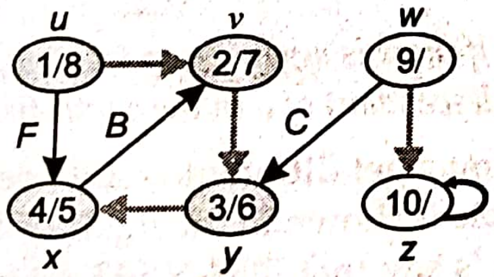
(h)

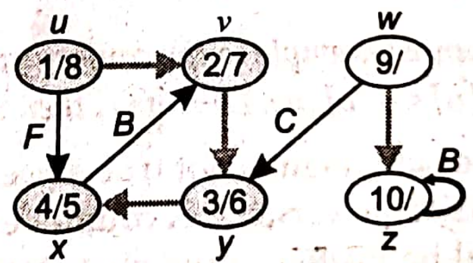

u
v
w
1/
2/7
F
B
4/5
3/6
x
y
z
(i)

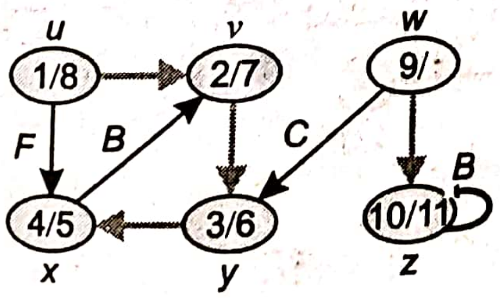

u
v
w
1/8
2/7
F
B
4/5
3/6
x
y
z
(j)

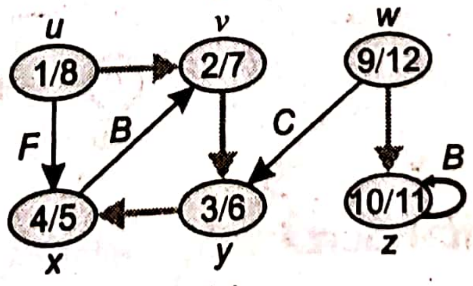

u
v
w
1/8
2/7
9/
F
B
4/5
3/6
x
y
z
(k)


u
v
w
1/8
2/7
9/
F
B
C
4/5
3/6
x
y
z
(l)


u
v
w
1/8
2/7
9/
F
B
C
4/5
3/6
10/
x
y
z
(m)


u
v
w
1/8
2/7
9/
F
B
C
B
4/5
3/6
10/
x
y
z
(n)


u
v
w
1/8
2/7
9/
F
B
C
B
4/5
3/6
10/11
x
y
z
(o)


u
v
w
1/8
2/7
9/12
F
B
C
B
4/5
3/6
10/11
x
y
z
(p)