Breadth First Search

Ms. Sasmita Kumari Nayak
Computer Science & Engineering

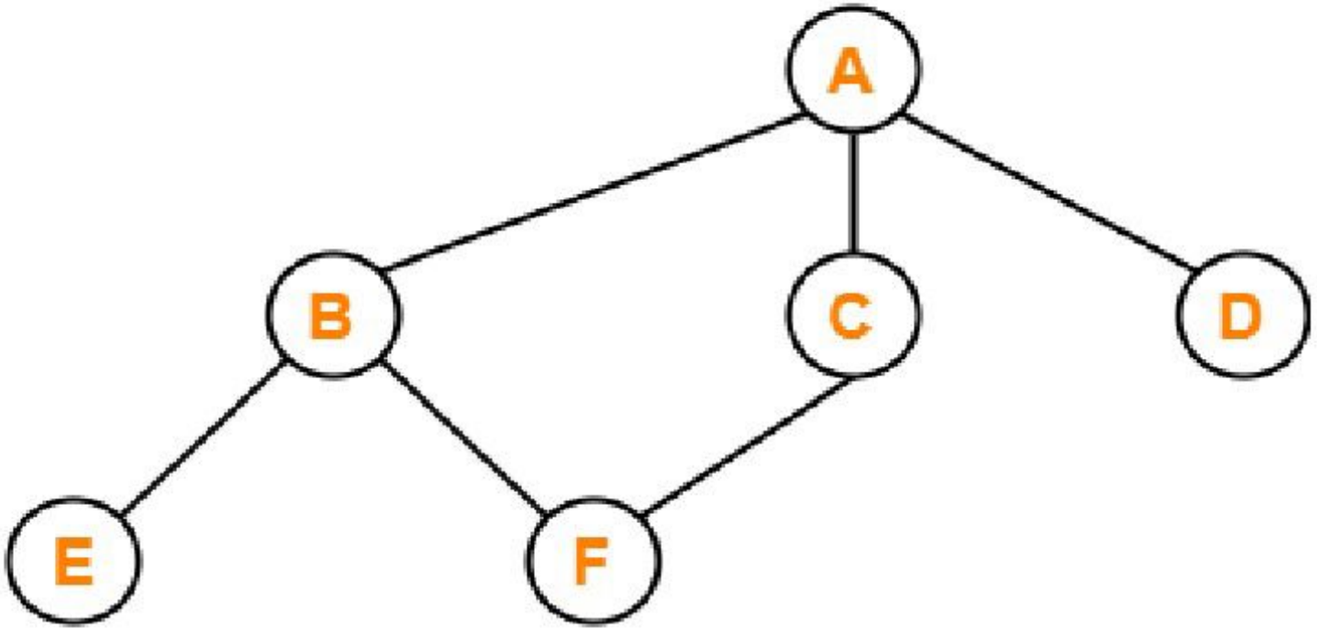
Breadth First Search (BFS)

- It is a graph traversal algorithm.
- It is used for traversing or searching a graph in a systematic fashion.
- Queue data structure is used in the implementation of breadth first search.

Cont…

- A node can be reached from different nodes using different paths but we need to visit each node only once.
- Each node have 3 categories
  - unvisited,
  - discovered and
  - complete.

Example



- The breadth first search traversal order of the above graph is-

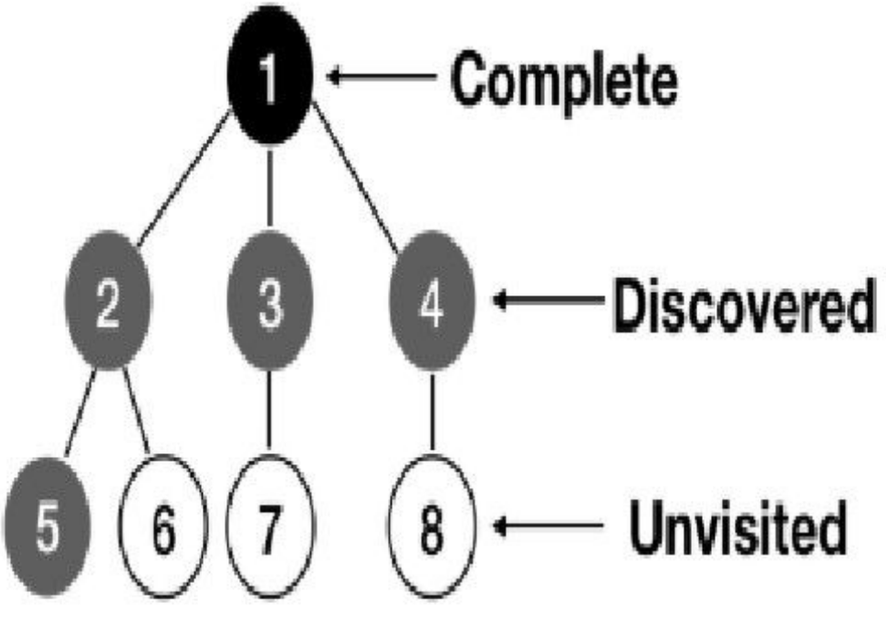
A, B, C, D, E, F

Working Procedure

- Initially, all nodes are unvisited. After visiting a node for the first time, it becomes discovered.
- A node is complete if all of its adjacent nodes have been visited.
- Thus, all the adjacent nodes of a complete node are either discovered or complete.

Cont…

- Generally, three different colors i.e., white, gray and black are used to represent unvisited, discovered and complete respectively.
- Thus after visiting a node, we first visit all its sibling and then their children.



- It uses a queue (FIFO) to achieve this. Starting from the source, we can put all its adjacent nodes in a queue.

Algorithm

```
BFS(G, s)                          // G is the graph and s is the starting node
   for each vertex u ∈ V [G] - {s}
       {
         do color[u] ← WHITE       // color of vertex u
         d[u] ← ∞                  // distance from source s to vertex u
         π[u] ← NIL                // predecessor of u
       }
   color[s] ← GRAY
   d[s] ← 0
   π[s] ← NIL
   Q ← Ø                           // Q is a FIFO - queue
   ENQUEUE(Q, s)
    while Q ≠ Ø                    // iterates as long as there are gray vertices. Lines 10-18
      {
        u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
             if color[v] = WHITE   // discover the undiscovered adjacent vertices
                {
              color[v] ← GRAY      // enqueued whenever painted gray
              d[v] ← d[u] + 1
              π[v] ← u
              ENQUEUE(Q, v)
                }
      color[u] ← BLACK             // painted black whenever dequeued
    }
```

Explanation

Step-01

- Create and maintain 3 variables for each vertex of the graph.
- For any vertex ‘v’ of the graph, these 3 variables are-
  - color[v]
  - Π[v]
  - d[v]

Cont…

1. color[v]

- This variable represents the color of the vertex v at the given point of time.
- The possible values of this variable are- WHITE, GREY and BLACK.
- WHITE color of the vertex signifies that it has not been discovered yet.
- GREY color of the vertex signifies that it has been discovered and it is being processed.
- BLACK color of the vertex signifies that it has been completely processed.

Cont…

- **2. Π[v]-**
- This variable represents the predecessor of vertex ‘v’.

- **3. d[v]-**
- This variable represents the distance of vertex v from the source vertex.

Cont…

Step-02

- For each vertex v of the graph except source vertex, initialize the variables as-
  - color[v] = WHITE
  - $\pi[v] = NIL$
  - $d[v] = \infty$

Step-3

- For source vertex S, initialize the variables as-
  - color[S] = GREY
  - $\pi[S] = NIL$
  - d[S] = 0

Cont…

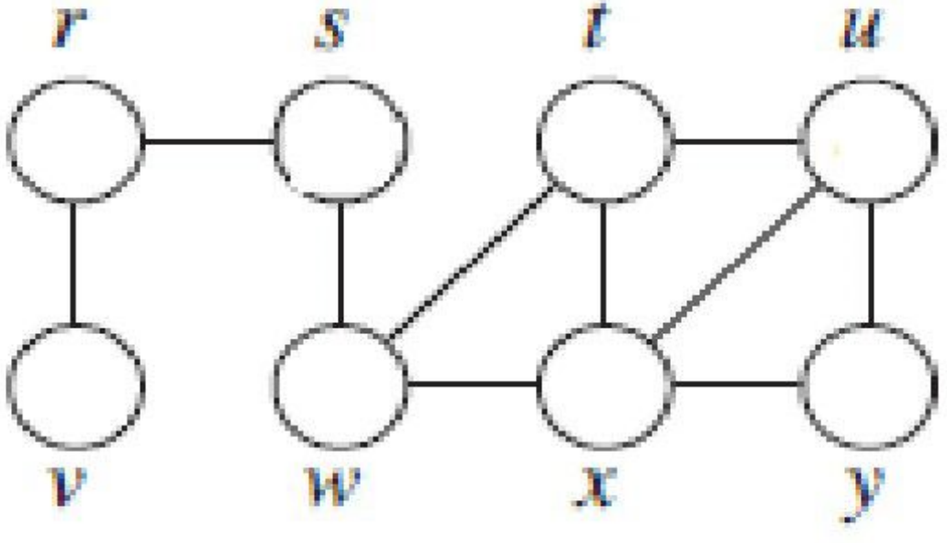
Step-4

- Now, enqueue source vertex S in queue Q and repeat the following procedure until the queue Q becomes empty-

  1. Dequeue vertex v from queue Q.
  2. For all the adjacent white vertices u of vertex v, do-
     - color[u] = GREY
     - d[u] = d[v] + 1
     - $\pi[u] = v$
     - Enqueue (Q,u)
  3. Color vertex v to black.

Example

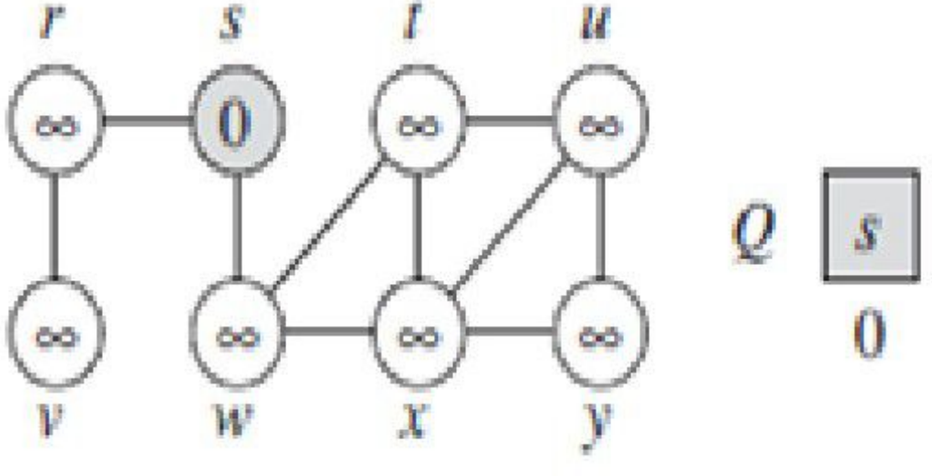
- Use the BFS algorithm to traverse the given graph.



Step-1

```
BFS(G, s)                          // G is the graph and s is the starting node
    for each vertex u ∈ V [G] - {s}
        {
          do color[u] ← WHITE      // color of vertex u
          d[u] ← ∞                 // distance from source s to vertex u
          π[u] ← NIL               // predecessor of u
        }
    color[s] ← GRAY
    d[s] ← 0
    π[s] ← NIL
    Q ← Ø                          // Q is a FIFO - queue
    ENQUEUE(Q, s)
```

u	r	s	t	u	v	w	x	y
π	N	N	N	N	N	N	N	N
C	W	G	W	W	W	W	W	W

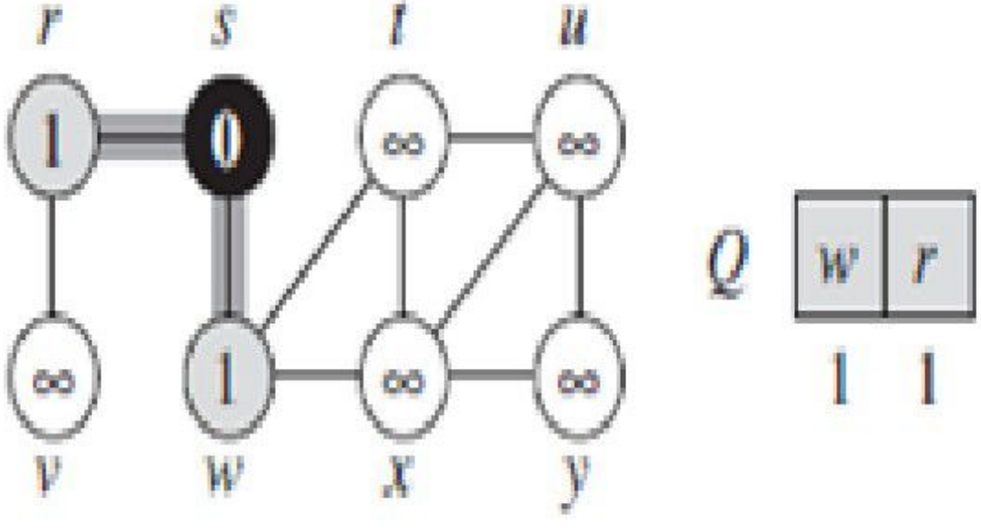


Step-2

- Dequeue vertex S from the Q
- For all adjacent white vertices ‘v’ (vertices R and W) of vertex S, we do-
  1. color[v] = GREY
  2. d[v] = d[S] + 1 = 0 + 1 = 1
  3. π[v] = S
  4. Enqueue all adjacent white vertices of S in queue Q
- color[S] = BLACK

```
while Q ≠ Ø
 {
    u ← DEQUEUE(Q)
    for each v ∈ Adj[u]
        if color[v] = WHITE
            {
            color[v] ← GRAY
            d[v] ← d[u] + 1
            π[v] ← u
            ENQUEUE(Q, v)
            }
   color[u] ← BLACK
}
```

u	r	s	t	u	v	w	x	y
π	s	N	N	N	N	s	N	N
C	G	B	W	W	W	G	W	W

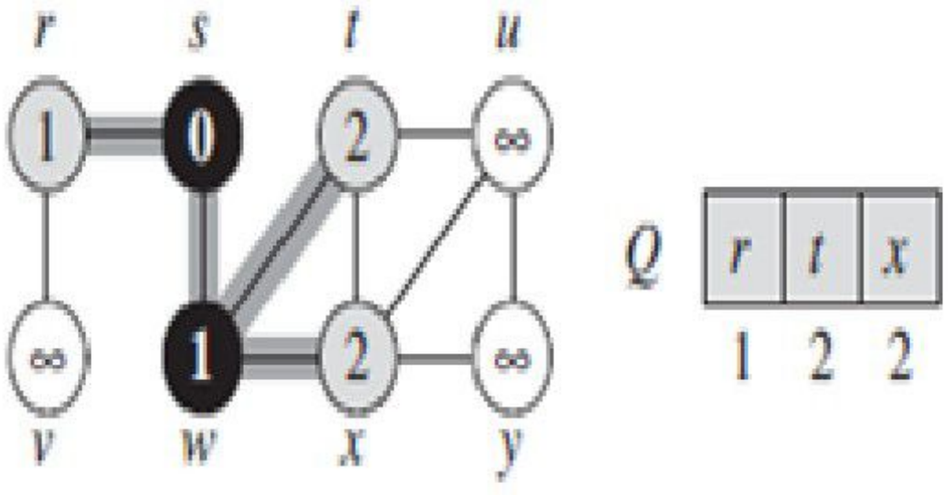


Cont…

- Continue the procedure till the queue will be empty.

Step-03:

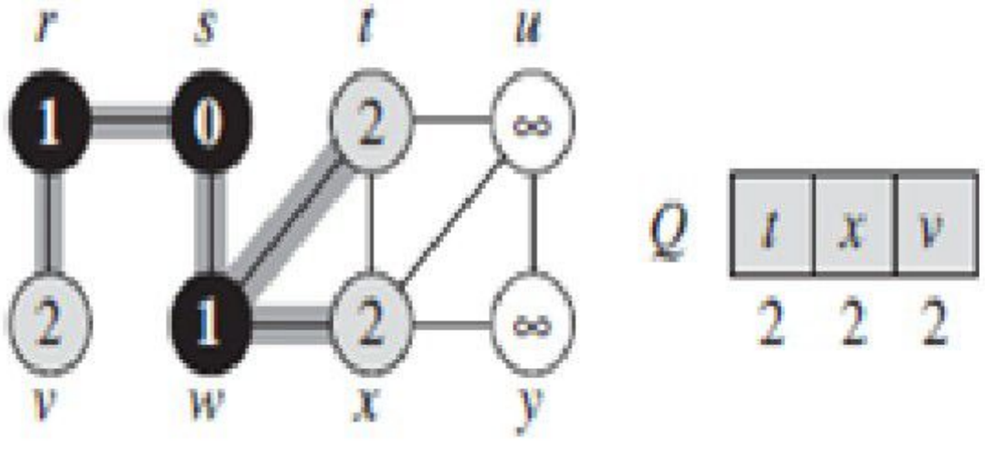
- Dequeue vertex w from the queue Q
- For all adjacent white vertices ‘v’ (vertices t and x) of vertex w, we do-
  1. color[v] = GREY
  2. d[v] = d[w] + 1 = 1 + 1 = 2
  3. π[v] = w
  4. Enqueue all adjacent white vertices of w in queue Q
- color[w] = BLACK



u	r	s	t	u	v	w	x	y
π	s	N	w	N	N	s	w	N
C	G	B	G	W	W	B	G	W

Step-4

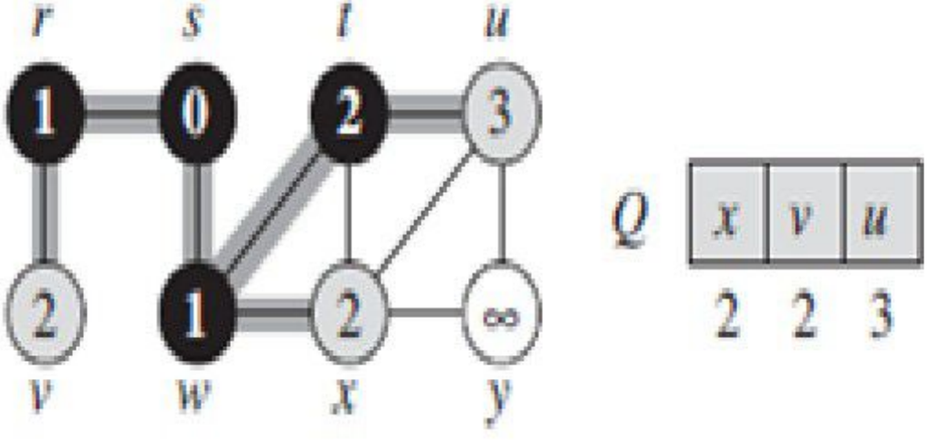
- Dequeue vertex r from the queue Q
- For all adjacent white vertices ‘v’ (vertex V) of vertex r, we do-
  1. color[v] = GREY
  2. d[v] = d[r] + 1 = 1 + 1 = 2
  3. π[v] = r
  4. Enqueue all adjacent white vertices of r in queue Q
- color[r] = BLACK



u	r	s	t	u	v	w	x	y
π	s	N	w	N	r	s	w	N
C	B	B	G	W	G	B	G	W

Step-5

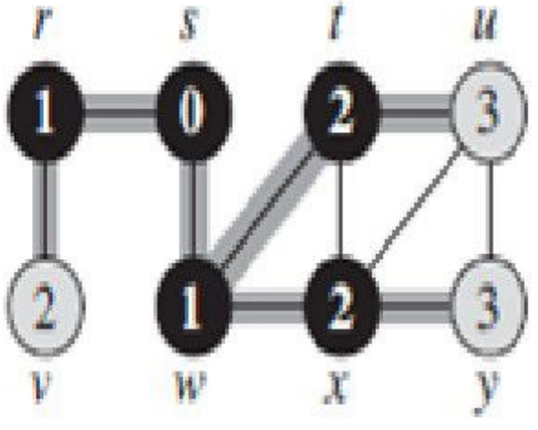
- Dequeue vertex t from the queue Q
- For all adjacent white vertices ‘v’ (vertex V) of vertex t, we do-
  1. color[v] = GREY
  2. d[v] = d[T] + 1 = 2 + 1 = 3
  3. π[v] = T
  4. Enqueue all adjacent white vertices of t in queue Q
- color[t] = BLACK



u	r	s	t	u	v	w	x	y
π	s	N	w	t	r	s	w	N
C	B	B	B	G	G	B	G	W

Step-6

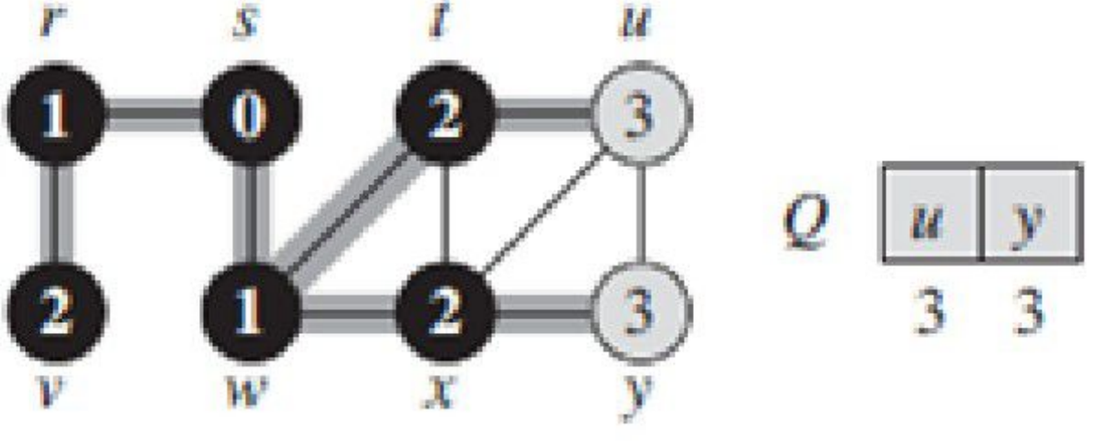
- Dequeue vertex x from the queue Q
- For all adjacent white vertices ‘v’ (vertex Y) of vertex x, we do-
  1. color[v] = GREY
  2. d[v] = d[x] + 1 = 2 + 1 = 3
  3. π[v] = x
  4. Enqueue all adjacent white vertices of x in queue Q
- color[x] = BLACK



u	r	s	t	u	v	w	x	y
π	s	N	w	t	r	s	w	x
C	B	B	B	G	G	B	B	G

Step-7

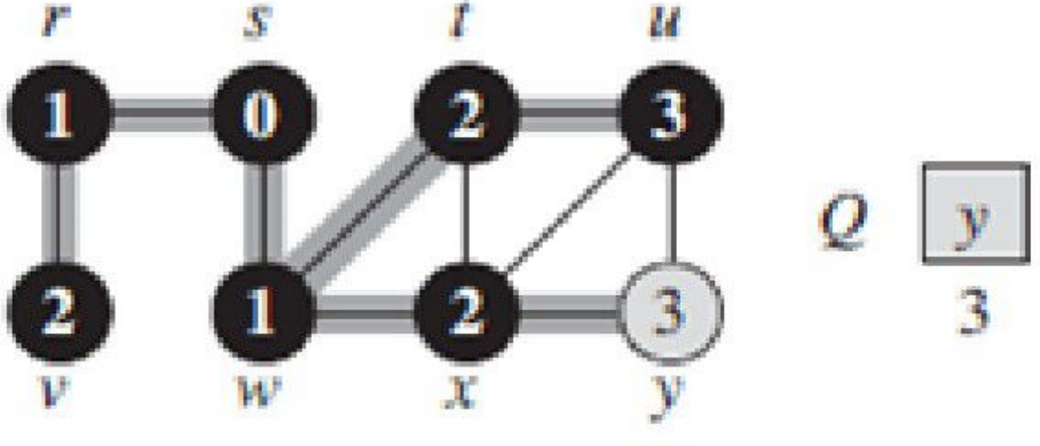
- Dequeue vertex v from the queue Q
- There are no adjacent white vertices to vertex v.
- Color[v] = BLACK



u	r	s	t	u	v	w	x	y
π	s	N	w	t	r	s	w	x
C	B	B	B	G	B	B	B	G

Step-8

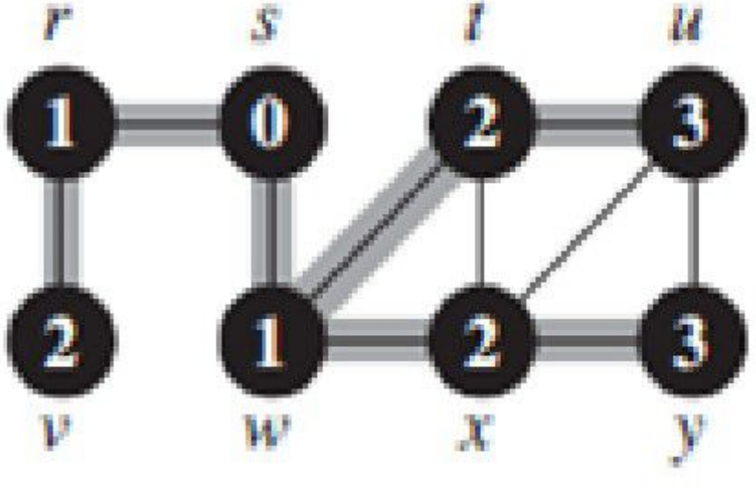
- Dequeue vertex U from the queue Q
- There are no adjacent white vertices to vertex U.
- color[U] = BLACK



u	r	s	t	u	v	w	x	y
π	s	N	w	t	r	s	w	x
C	B	B	B	B	B	B	B	G

Step-9

- Dequeue vertex Y from the queue Q
- There are no adjacent white vertices to vertex Y.
- color[Y] = BLACK
- Since, all the vertices have turned black and the queue has got empty, so we stop.

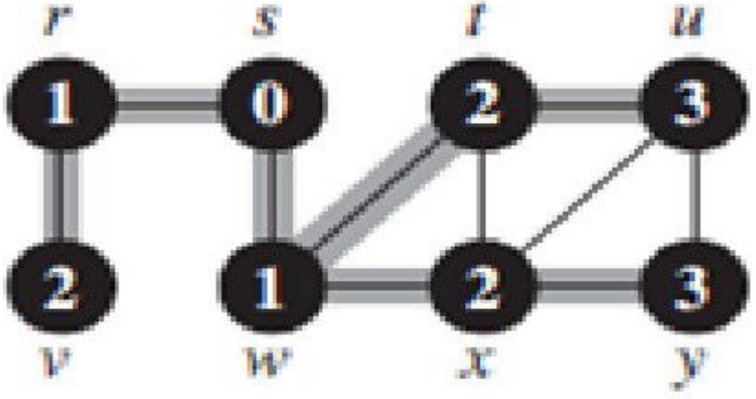


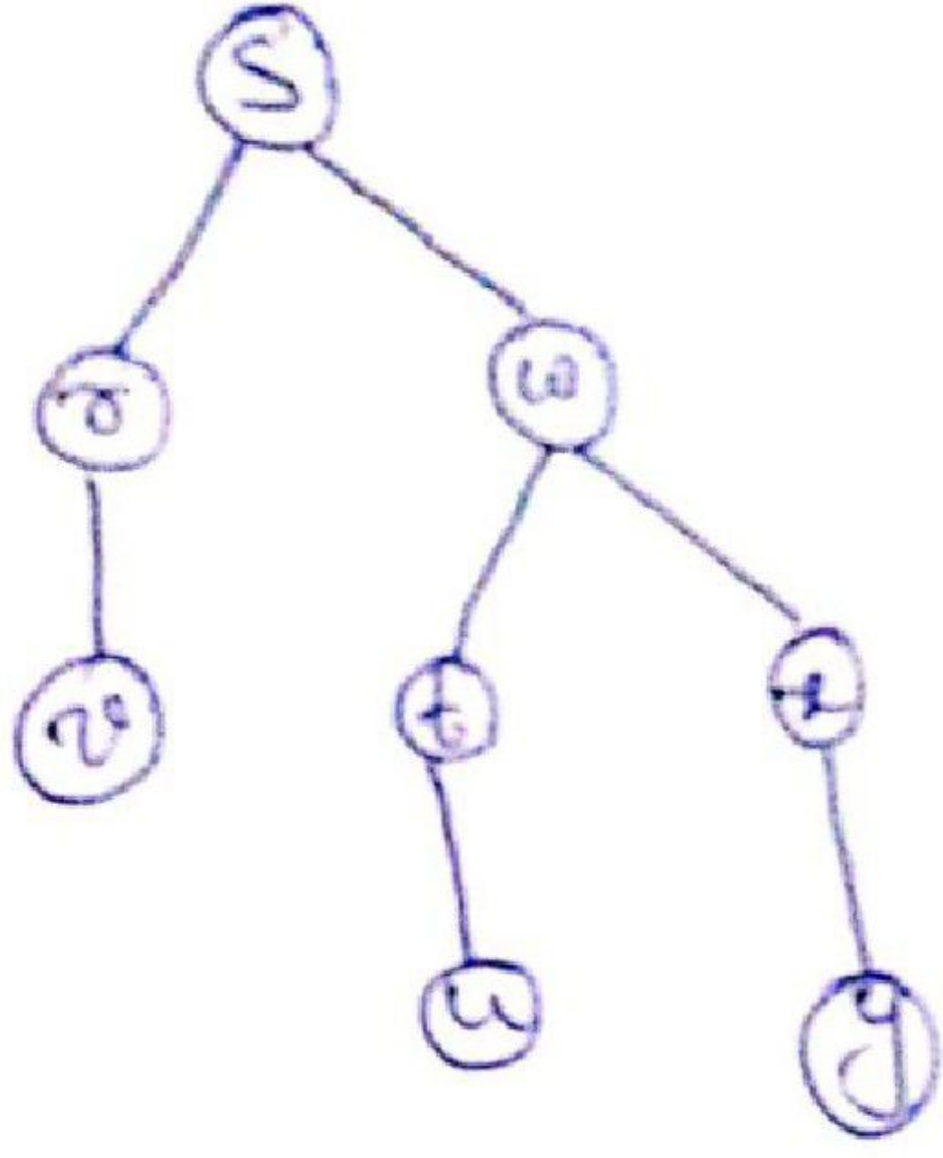
$Q \quad \emptyset$

u	r	s	t	u	v	w	x	y
π	s	N	w	t	r	s	w	x
C	B	B	B	B	B	B	B	B

Result

- In BFS, the result will be a tree.

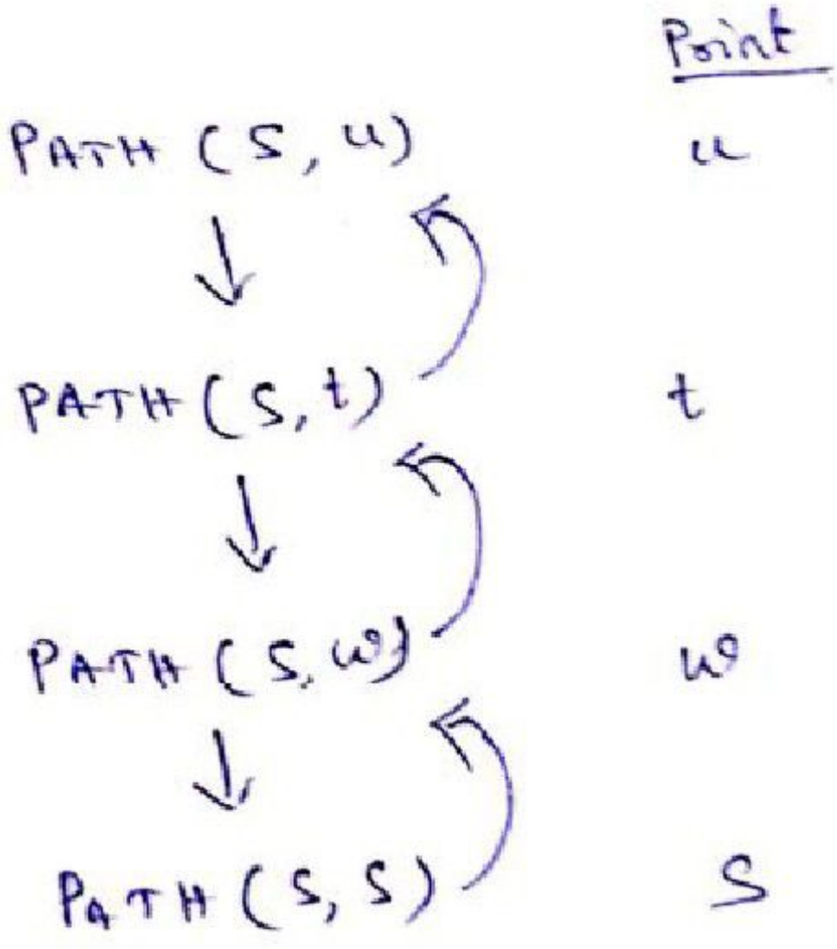




u	r	s	t	u	v	w	x	y
π	s	N	w	t	r	s	w	x
C	B	B	B	B	B	B	B	B

Find the path from s to u

```
PRINT-PATH(G, s, v)
1  if v == s
2      print s
3  elseif π[v] == NIL
4      print "no path from" s "to" v "exists"
5  else PRINT-PATH(G, s, π[v])
6      print v
```



Path from s to u is:

s w t u Any.

BFS Time Complexity

- The total running time for BFS is O (V+E).
- The running time of BFS algorithm is =

  running time for initialization

  +

  running time for adjacency

  = i.e. O (V+E).

- The running time of BFS algorithm is upto the number of vertices time, which is a linear time. In summery the entire algorithm will run in O (V+E).

Thank You