Data Structures and Algorithms

Breadth-First Search
Depth-First Search

Prepared By: Dr. Waheed Iqbal
Modified By: Abdul Mateen

Breadth-First Search (BFS)

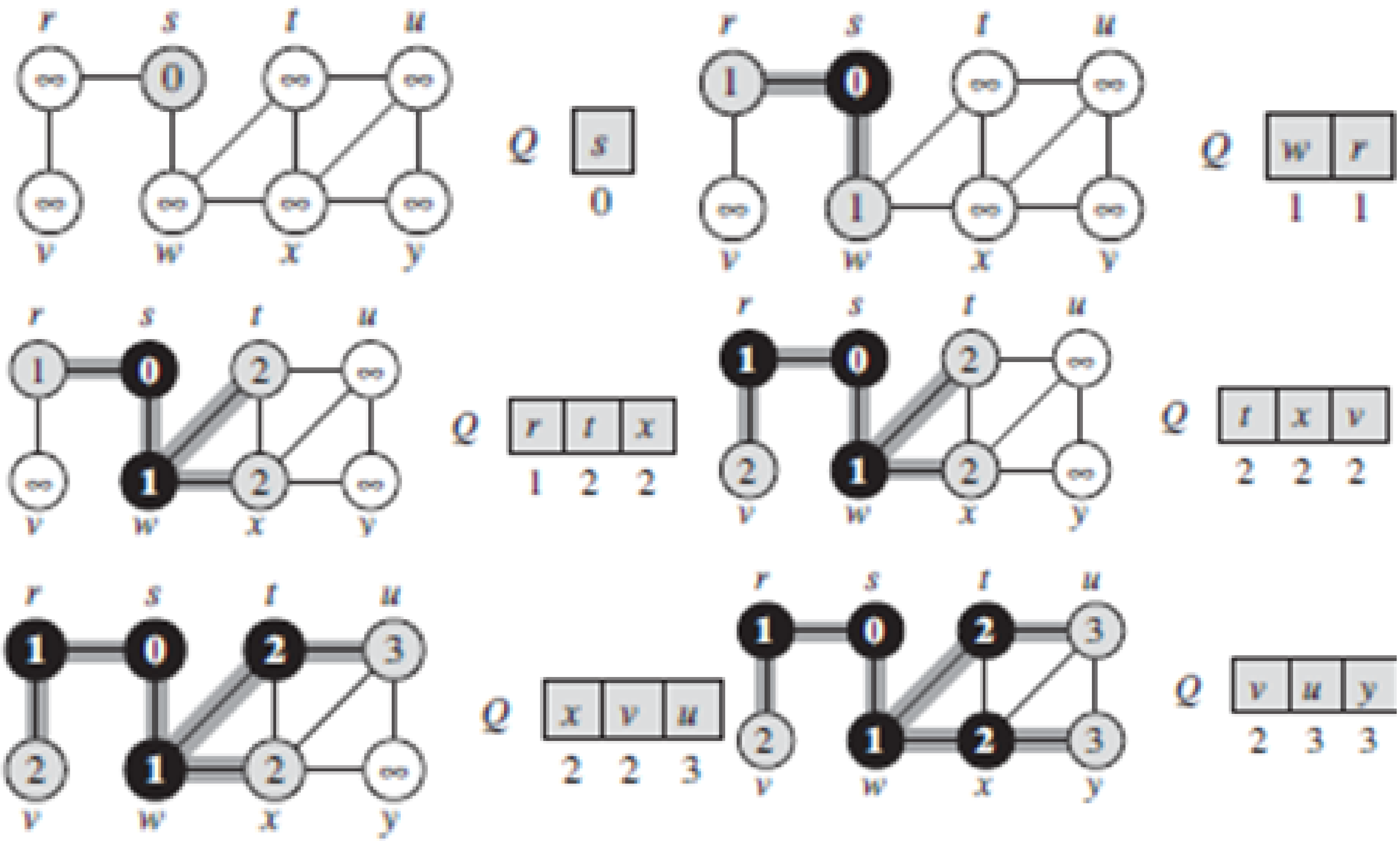
- One of the simplest algorithm for searching a graph
- Given a graph $G = (V, E)$ and a distinguished ***source vertex*** $\boldsymbol{s}$, ***breadth-first*** search systematically explores the edges of G to "discover" every vertex that is reachable from s
- It computes the distance (smallest number of edges) from s to each reachable vertex

Breadth-First Search (Cont.)

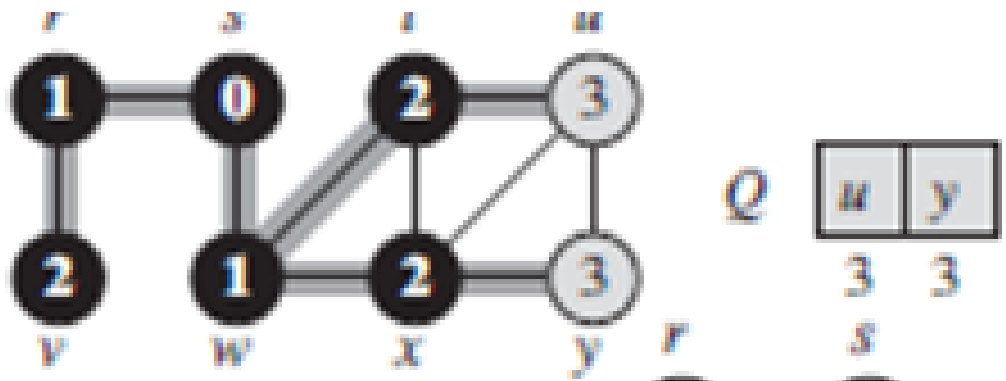
Algorithm

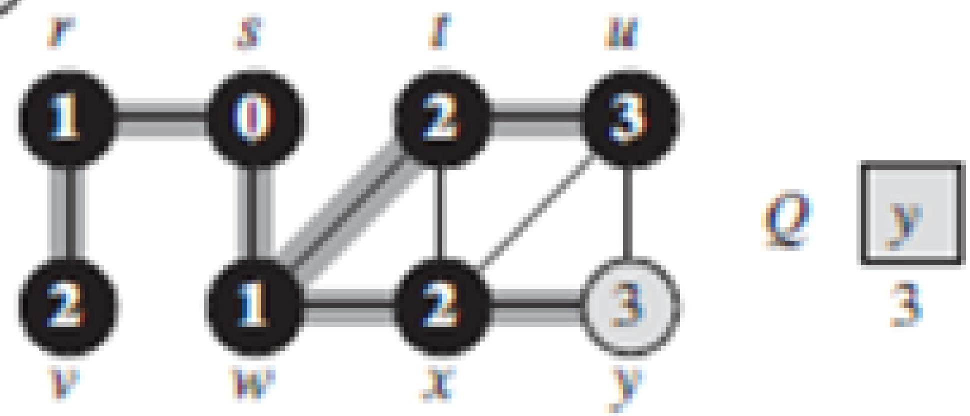
```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

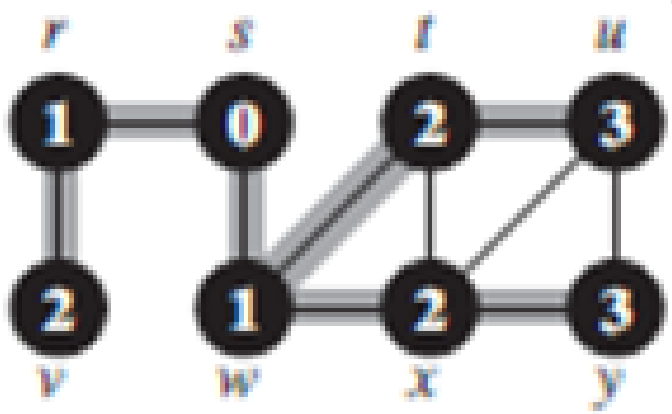
Breadth-First Search (Cont.)



Breadth-First Search (Cont.)







Breadth-First Search (Cont.)

- Enqueuing and dequeuing take O(1)
- Total time devoted to queue operations take O(V)
- Total time scanning adjacency lists is O(E)
- Total running time of the BFS procedure is O(V +E)

Breadth-First Search (Cont.)

Shortest Path

- The procedure BFS builds a breadth-first tree as it searches the graph

- Shortest-path from ***s*** to ***v*** as the minimum number of edges in any path from vertex ***s*** to vertex ***v***;

```
PRINT-PATH(G, s, v)
if v == s
    print s
elseif v.π == NIL
    print "no path from" s "to" v "exists"
else PRINT-PATH(G, s, v.π)
    print v
```

Applications

1. **Shortest Path in Unweighted Graphs**
2. **Finding Connected Components**
3. **Cycle Detection in Undirected Graphs**
4. **Level Order Traversal of Trees**
5. **Web Crawlers**
6. **Social Networking Sites**
7. **Broadcasting in Networks**
8. **Maze and Puzzle Solving**

Depth-first Search (DFS)

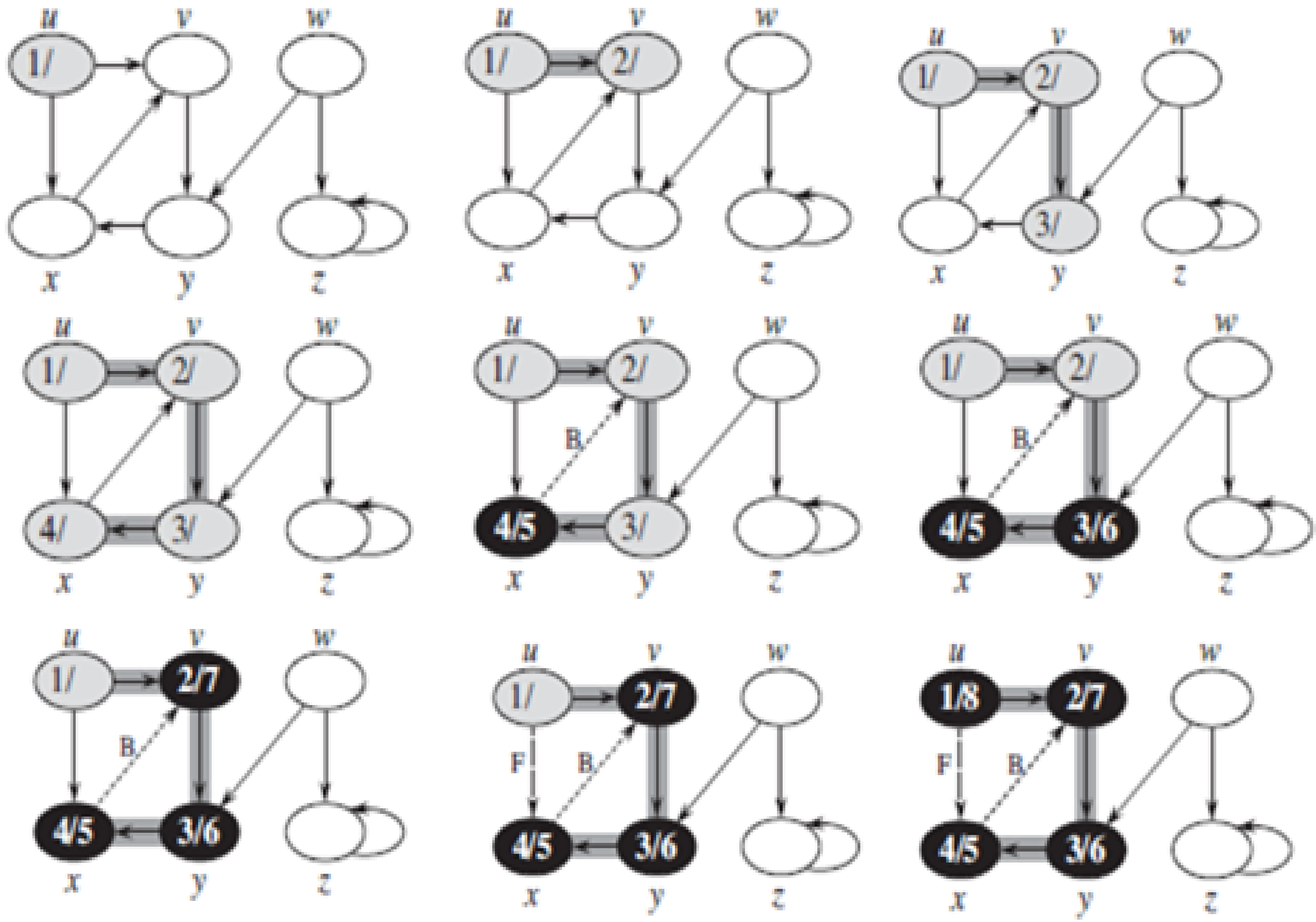
- Depth-first search explores edges out of the most recently discovered vertex that still has unexplored edges leaving it.

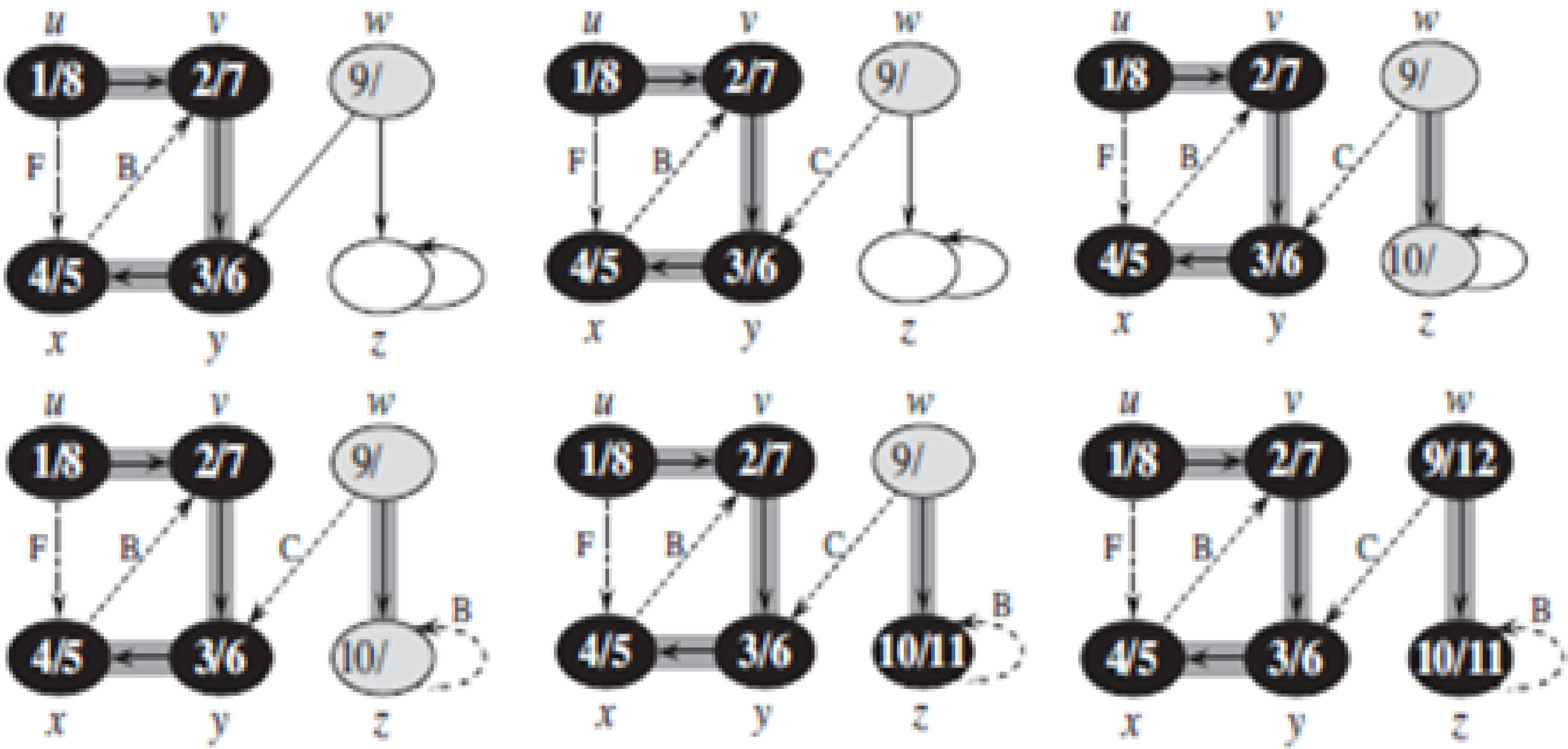
- Once all of *v*'s edges have been explored, the search "backtracks" to explore edges leaving the vertex from which was discovered.

- This process continues until we have discovered all the vertices that are reachable from the original source vertex.

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

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


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



- DFS is O(V) exclusive of DFS-VIST
- DFS-VISIT is O(E)
- The running time of DFS is therefore O(V+E)

Applications

1. **Path Finding**
2. **Topological Sorting**
3. **Cycle Detection**
4. **Strongly Connected Components**
5. **Solving Mazes and Puzzles**
6. **Graph Coloring**
7. **Artificial Intelligence**
8. **Network Analysis**
9. **Web Crawlers**