Data Structures & Algorithms

11: Graph Searching - BFS

Dr Ram Prasad Krishnamoorthy

Associate Professor
School of Computing and Data Science

ram.krish@saiuniversity.edu.in

SAI UNIVERSITY

Graph Representations

Graph Searching - BFS

Given graph **G = (V, E)**.

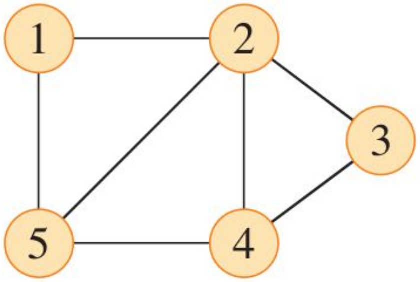
In pseudocode, represent vertex set by **G.V** and edge set by **G.E**.

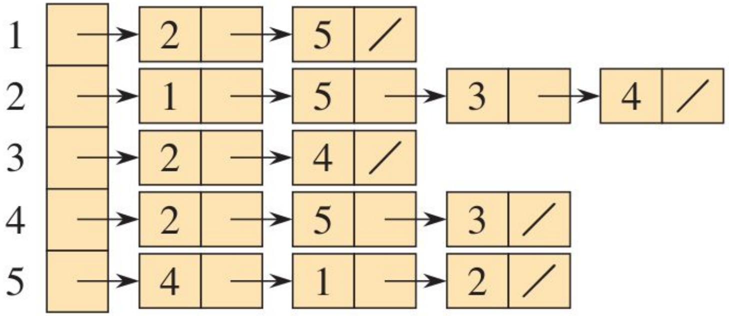
G may be either **directed** or **undirected**.

Two common ways to represent graphs for algorithms:

1. Adjacency Lists.
2. Adjacency Matrix.

Graph Searching - BFS



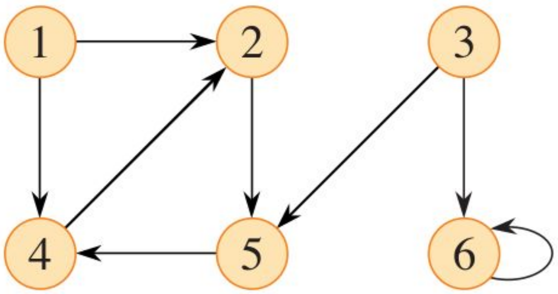


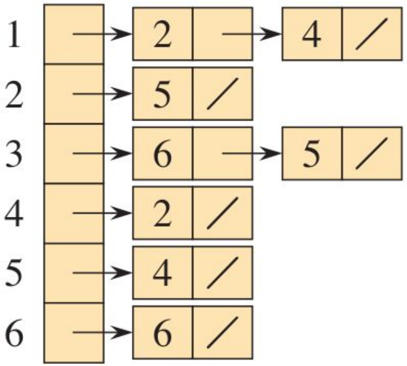
Adjacency List

	1	2	3	4	5
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

Adjacency Matrix

Graph Searching - BFS





Adjacency List

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

Adjacency Matrix

Graph Searching - BFS

Adjacency lists

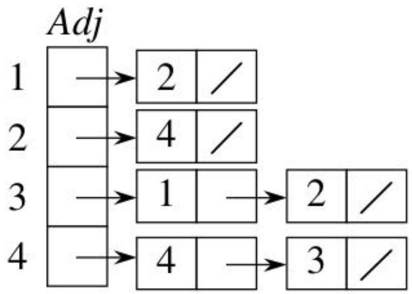
Array ***Adj*** of **|V|** lists, one per *vertex*.

Vertex ***u***'s list has all vertices ***v*** such that ***(u, v)*** $\in$ ***E***.

(Works for both directed and undirected graphs.)

In pseudocode, denote the array as attribute ***G.Adj***.

We will see notation such as ***G.Adj[u]***



Adjacency matrix

$$|V| \times |V| \text{ matrix } A = (a_{ij})$$

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \,, \\ 0 & \text{otherwise} \,. \end{cases}$$

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

Graph Searching

Graph Searching - BFS

Graph Searching

Searching a graph means systematically following the edges of the graph so as to visit the vertices of the graph.

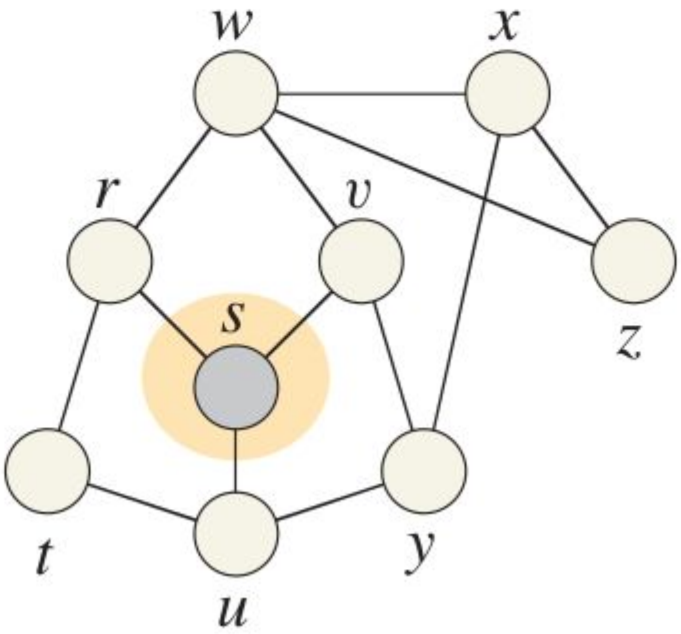
Graph searching algorithms discovers the structure of the graph.

Two prominent Graph Searching

- Breadth First Search(BFS)
- Depth First Search (DFS)

Breadth First Search (BFS)

Graph Searching - BFS



Graph Searching - BFS

Breadth First Search (BFS)

BFS, from a given source vertex **s**, finds the shortest simple path to vertices.

$\Rightarrow$ discovers every vertex **v** reachable from **s** through a *shortest path*.

$\rightarrow$ **shortest path** is the path containing *smallest number of edges*.

BFS on a graph with source vertex **s**, generate a **Breadth First Tree** with **s** as root.

Breadth First $\Rightarrow$ A wave (also mentioned as frontiers) emanating from source **s**, visiting vertices at **distance 1**, then **distance 2** etc until it has discovered every vertex reachable from **s**.

Graph Searching - BFS

Breadth First Search (BFS)

To keep track of the wave, BFS uses a Queue.

$\Rightarrow$ Queue contains **some vertices** at distance **k** and,

possibly **some vertices** at distance **k+1**.

Queue contains vertices belonging to *portions* of **two consecutive waves** at any time.

Graph Searching - BFS

Breadth First Search (BFS)

Colouring scheme in BFS:

BFS colors vertices **WHITE**, **GRAY** and **BLACK**.

- **WHITE** → not visited, and not reachable from s
- **GRAY** → discovered first time a vertex is reachable from s. *(contained in queue)*
- **BLACK** → once all of the vertex's edges are explored.

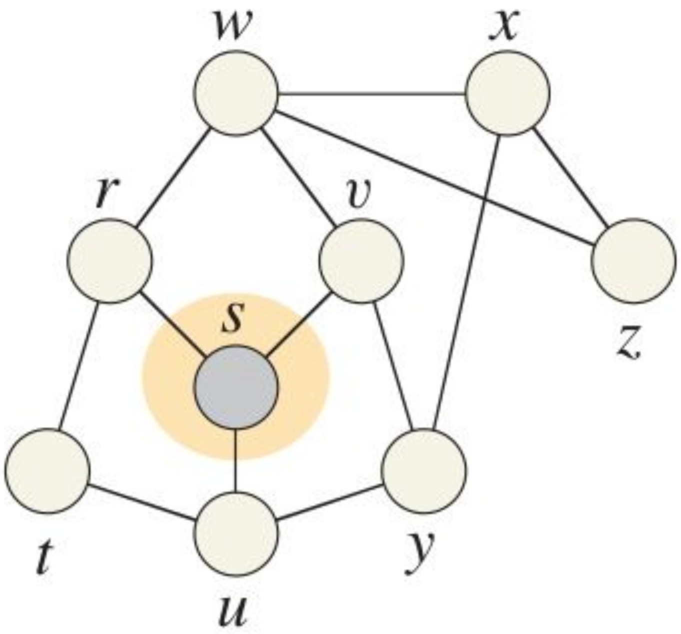
Graph Searching - BFS

Breadth First Search (BFS)

Additional attributes in to each vertex *v*:

- $v.color$ is the color of v: WHITE, GRAY, or BLACK.
- $v.d$ holds the distance from the source vertex s to v, as computed by the algorithm.
- $v.\pi$ is v's predecessor in the breadth-first tree. If v has no predecessor because it is the source vertex or is undiscovered, then $v.\pi = \text{NIL}$.

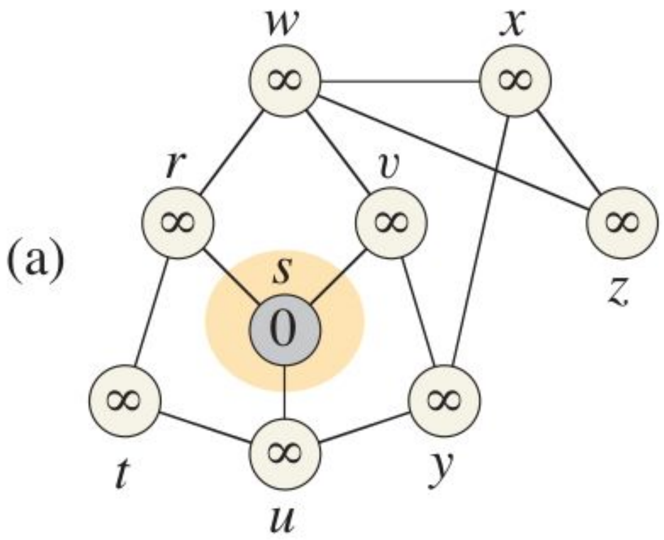
Graph Searching - BFS



Graph Searching - BFS

```
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 vertex v in G.Adj[u]
            if v.color == WHITE
                v.color = GRAY
                v.d = u.d + 1
                v.π = u
                ENQUEUE(Q, v)
        u.color = BLACK
```

Graph Searching - BFS



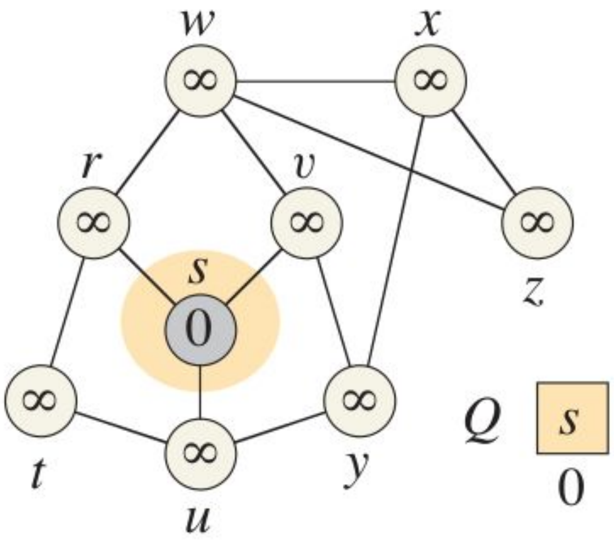
Graph Searching - BFS

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 vertex v in G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

Graph Searching - BFS

(a)



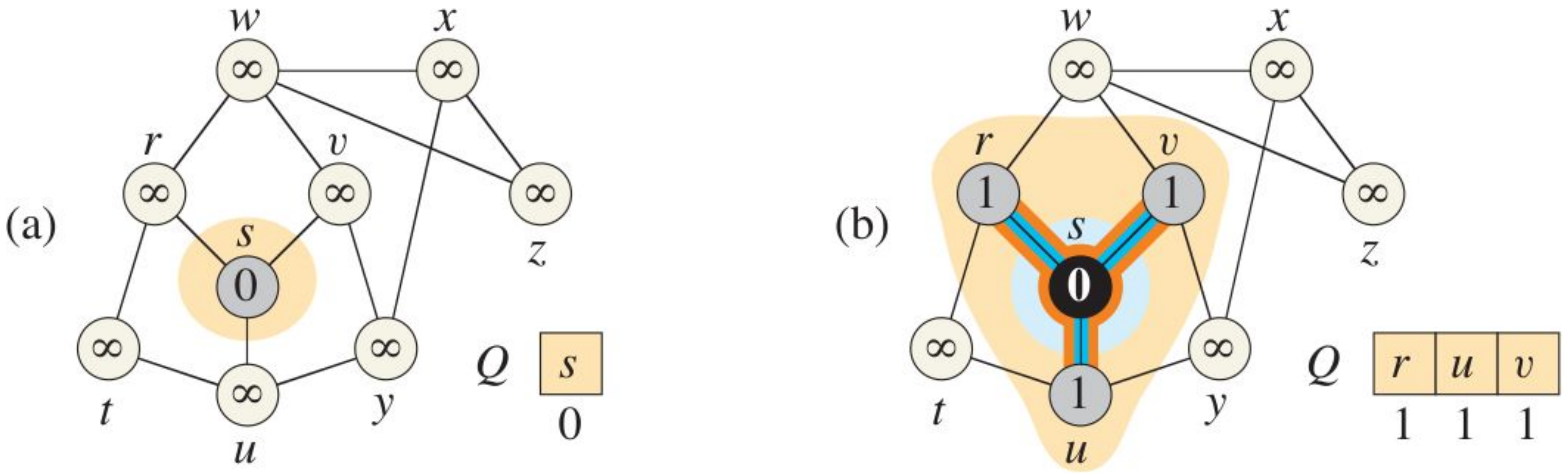
Graph Searching - BFS

BFS(G, s)

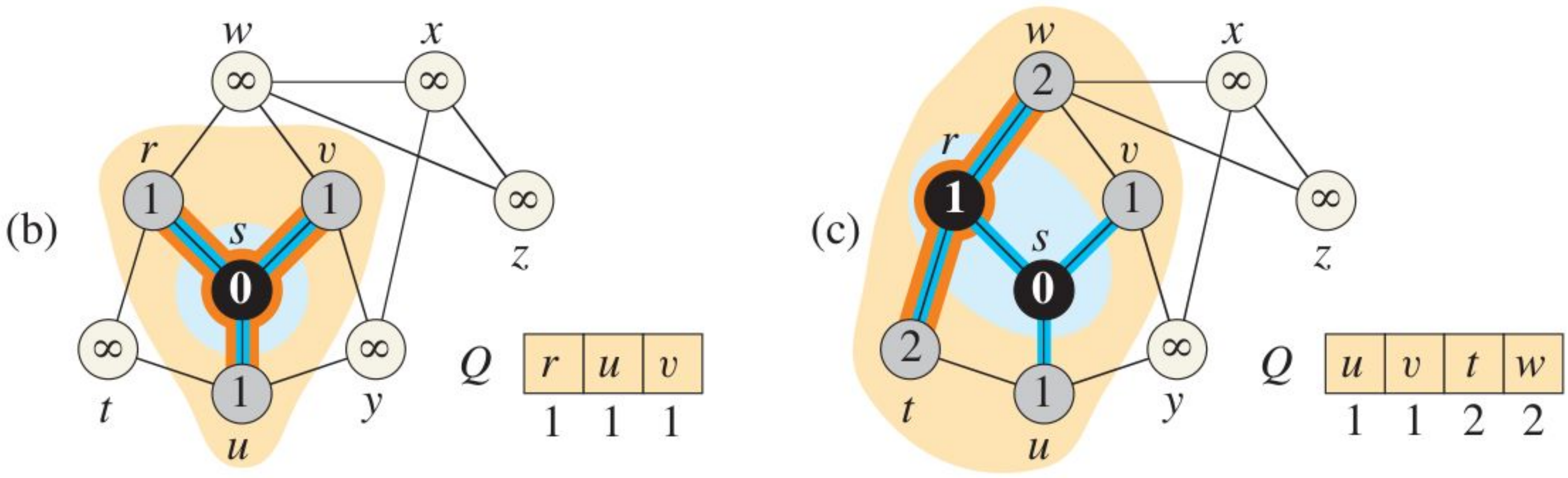
```
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
```

```
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each vertex v in G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

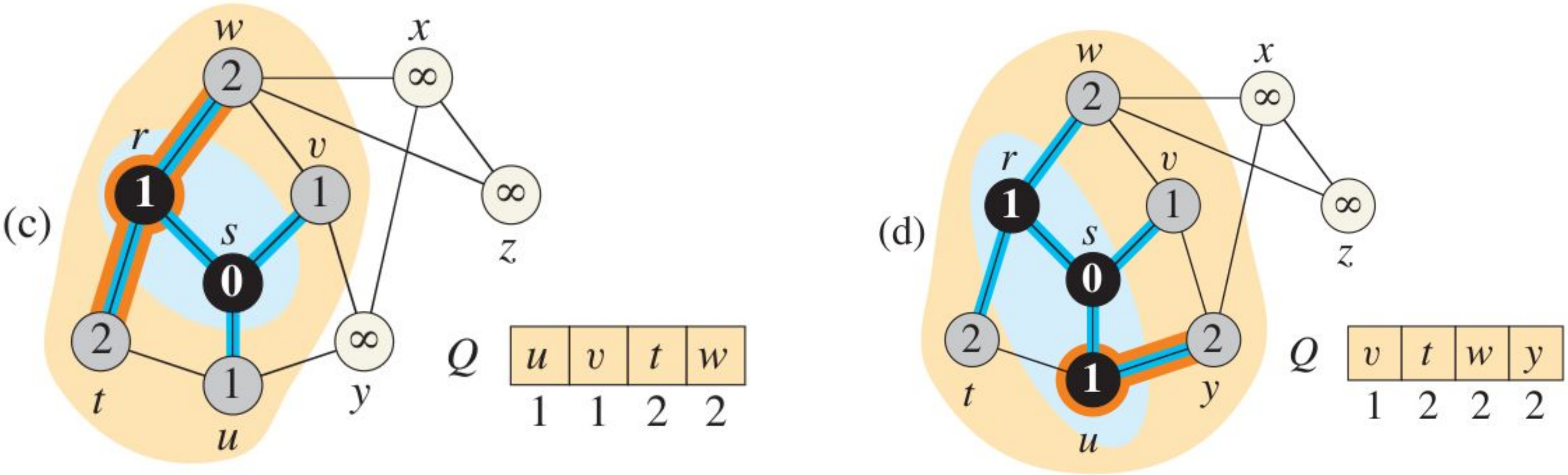
Graph Searching - BFS



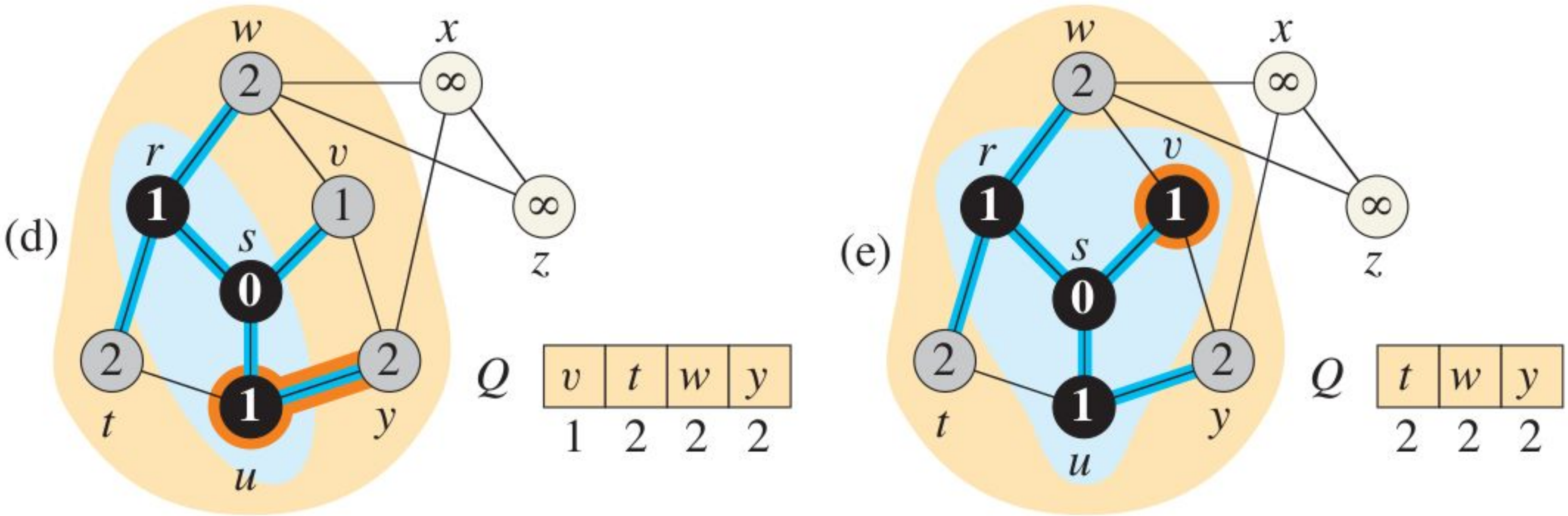
Graph Searching - BFS



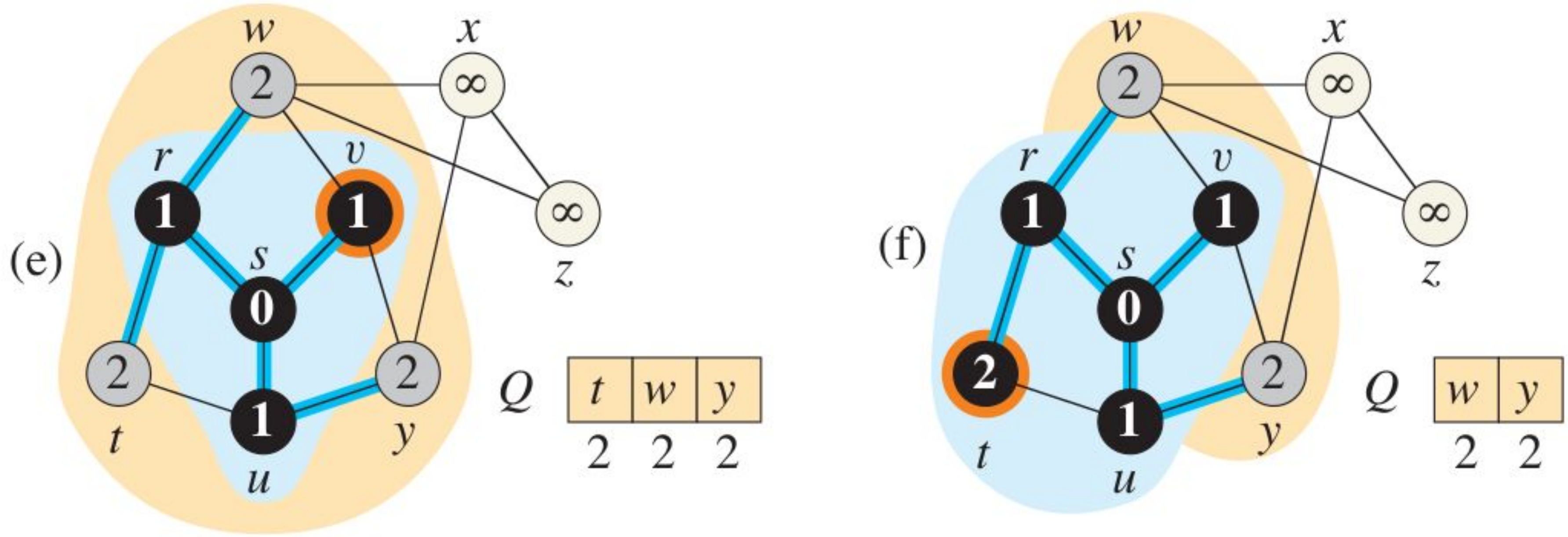
Graph Searching - BFS



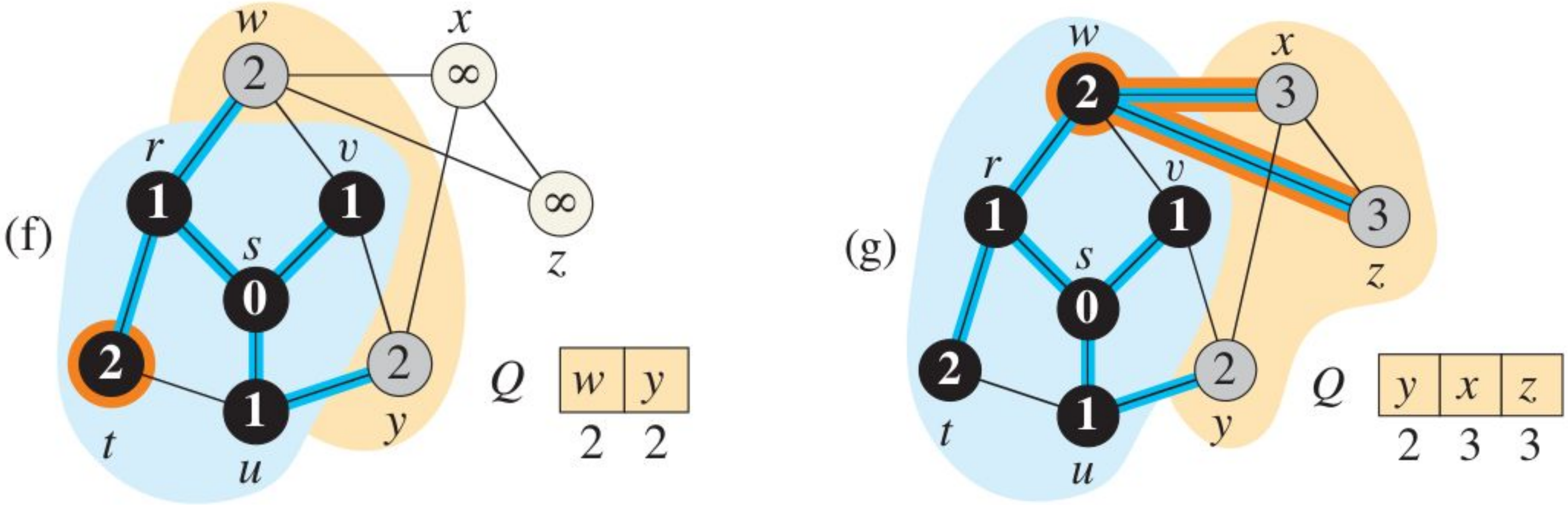
Graph Searching - BFS



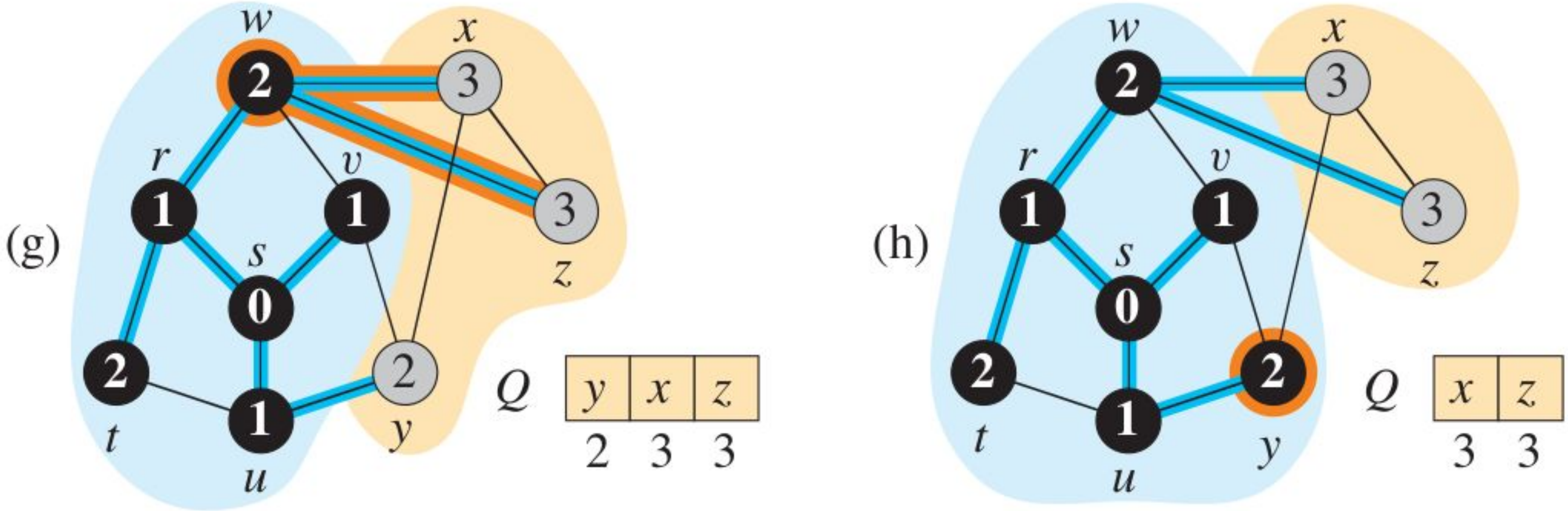
Graph Searching - BFS



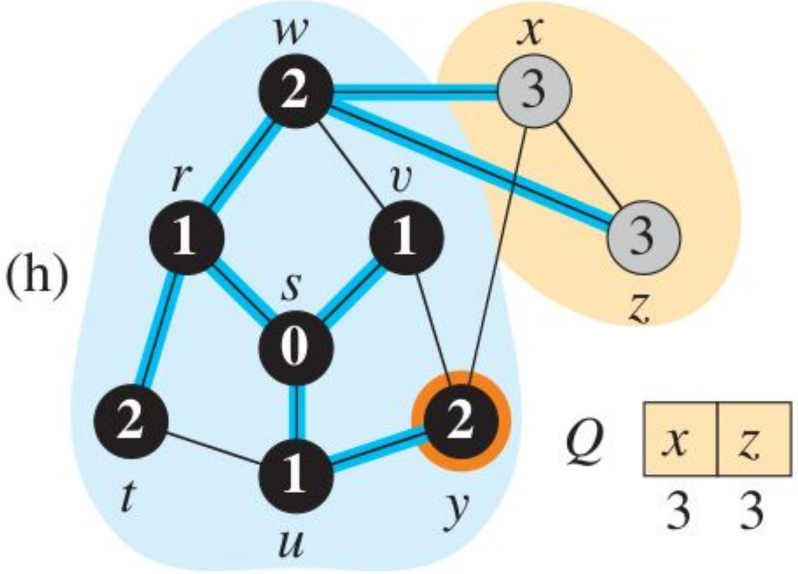
Graph Searching - BFS

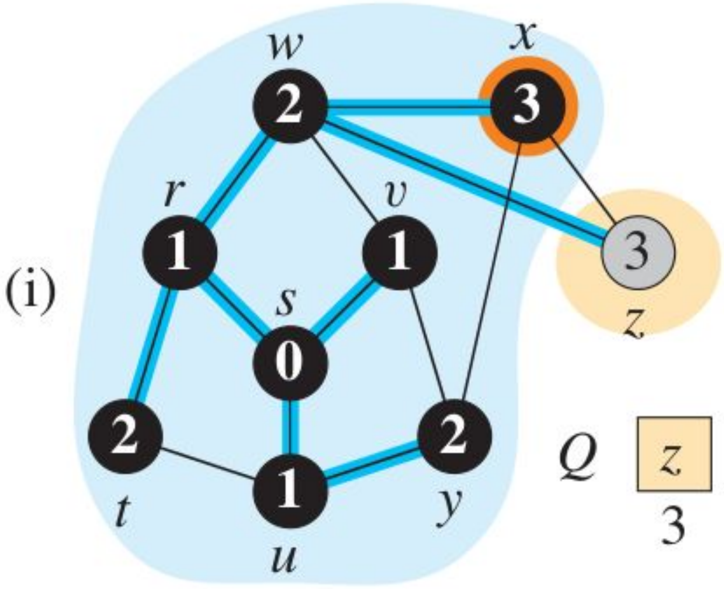


Graph Searching - BFS



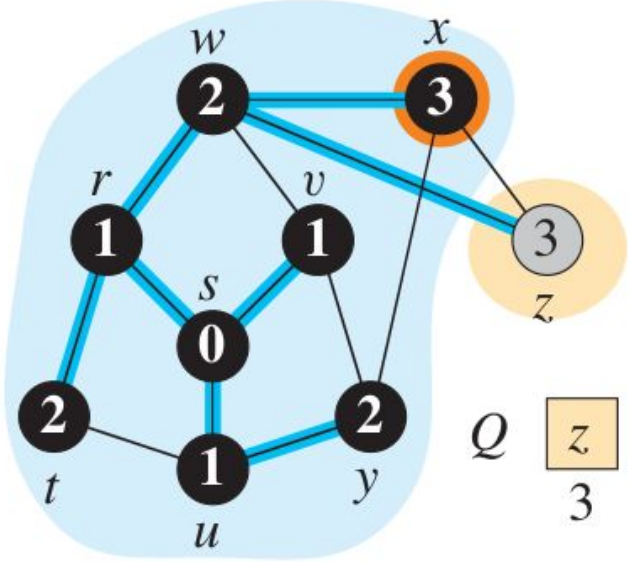
Graph Searching - BFS





Graph Searching - BFS

(i)



(j)