Graphs: Breadth First Search

Bijendra Nath Jain

bnjain@iiitd.ac.in

Some of the slides are from https://courses.cs.washington.edu/courses/cse373/22sp/

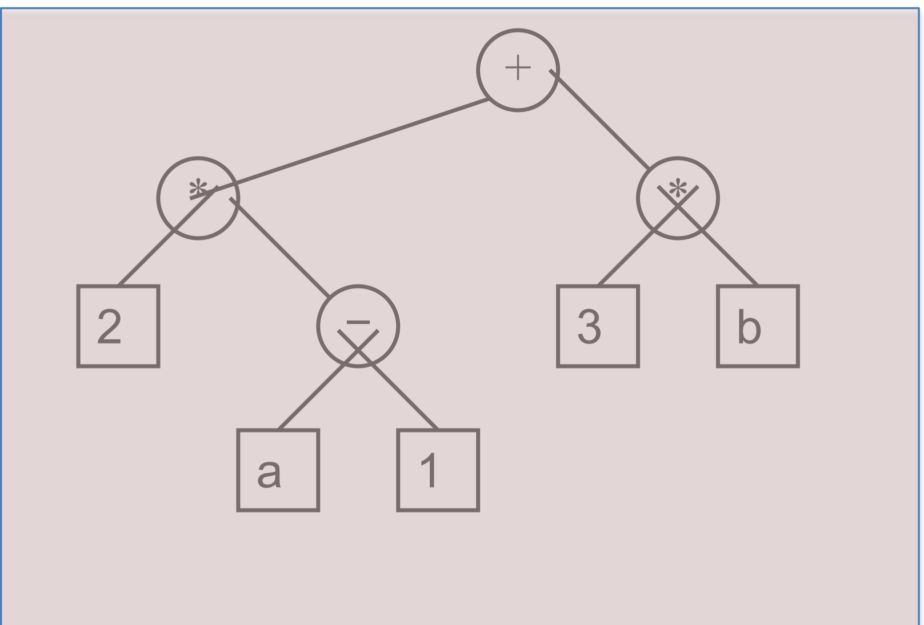
Outline

- Graphs:
  - Undirected graphs
  - Directed graphs
  - (Directed) acyclic graphs (or DAGs)
  - Sparse graphs
  - Weighted graphs
- Graph applications
- Representation of graphs:
  - Adjacency matrix
  - Linked lists
- Algorithms:
  - Traversal algorithms:
    - BFS
    - DFS
  - Topological sort
  - Minimum spanning trees
  - Dijkstra's Shortest path
    - One-to-one
    - One-to-many
    - Many-to-many

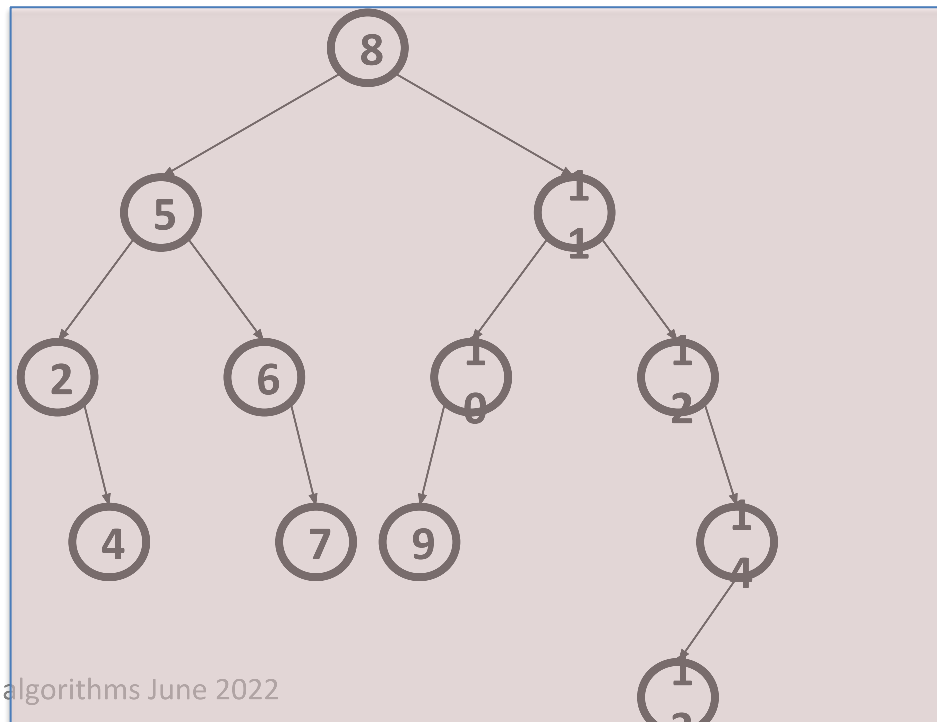
Traversal of graphs: Breadth-First Search, or BFS

- Traversal of graphs:
  - Systematic way of "searching" through all vertices in a graph
  - Mainly two search methods
    - BFS
    - DFS
- Which one to use depends upon the end-application, as with binary tree traversals

Post-order traversal of binary tree to evaluate an arithmetic expression such as (2 * (a − 1) + (3 * b))



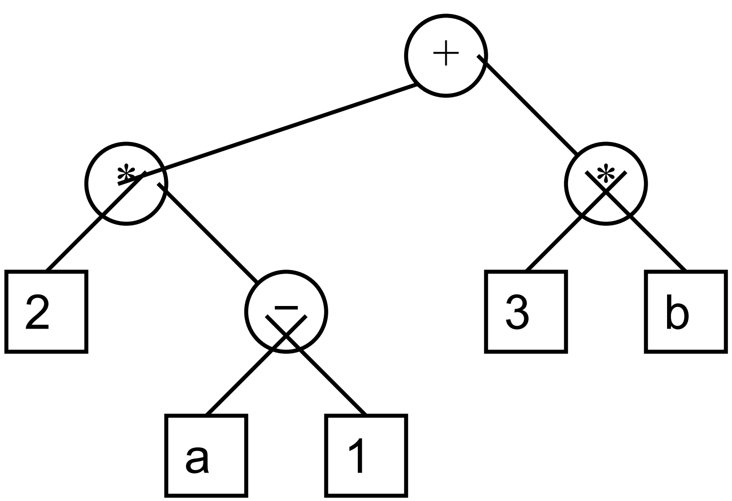
In-order traversal of binary search tree gives sorted list of objects



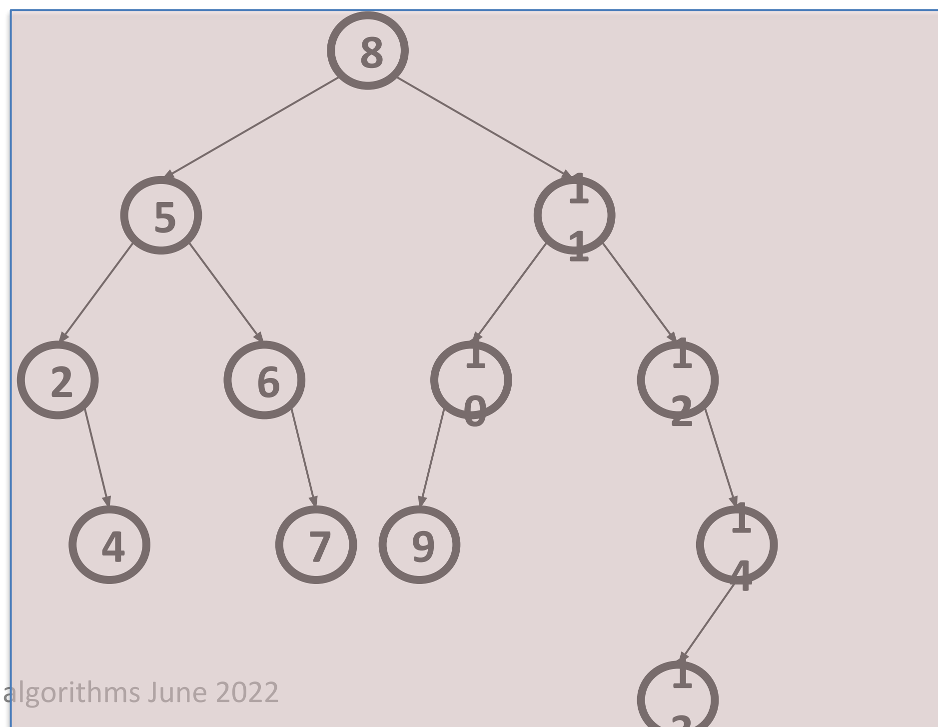
Traversal of graphs: Breadth-First Search, or BFS

- Traversal of graphs:
  - Systematic way of “searching” through all vertices in a graph
  - Mainly two search methods
    - BFS
    - DFS
- Which one to use depends upon the end-application, as with binary tree traversals

Post-order traversal of binary tree to evaluate an arithmetic expression such as (2 * (a − 1) + (3 * b))



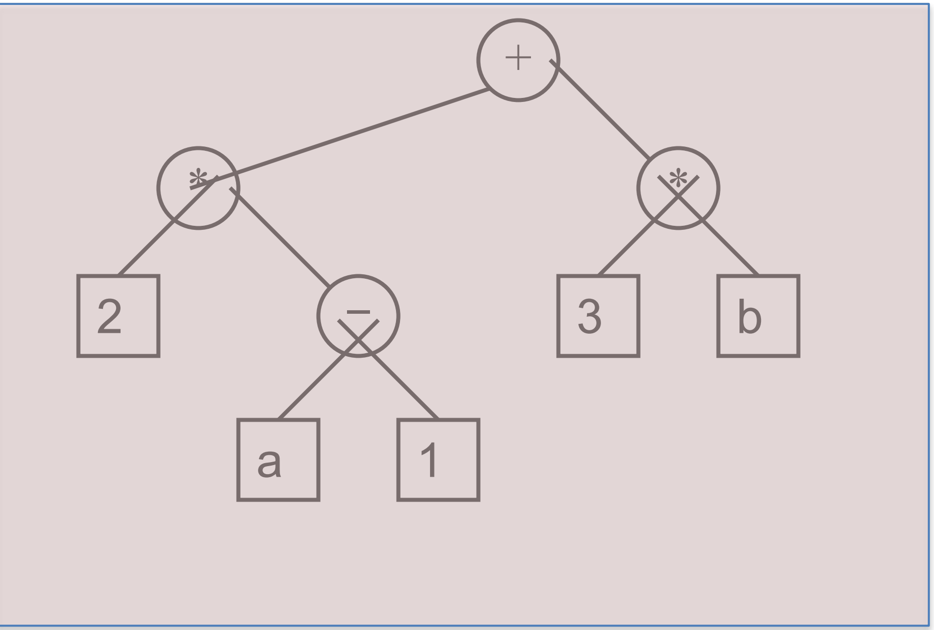
In-order traversal of binary search tree gives sorted list of objects



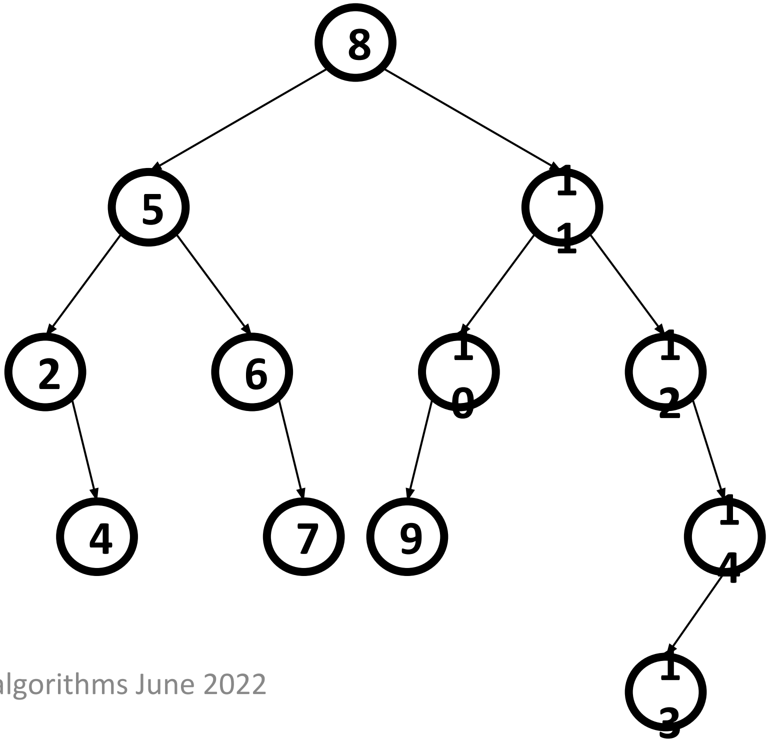
Traversal of graphs: Breadth-First Search, or BFS

- Traversal of graphs:
  - Systematic way of "searching" through all vertices in a graph
  - Mainly two search methods
    - BFS
    - DFS
- Which one to use depends upon the end-application, as with binary tree traversals

Post-order traversal of binary tree to evaluate an arithmetic expression such as (2 * (a − 1) + (3 * b))

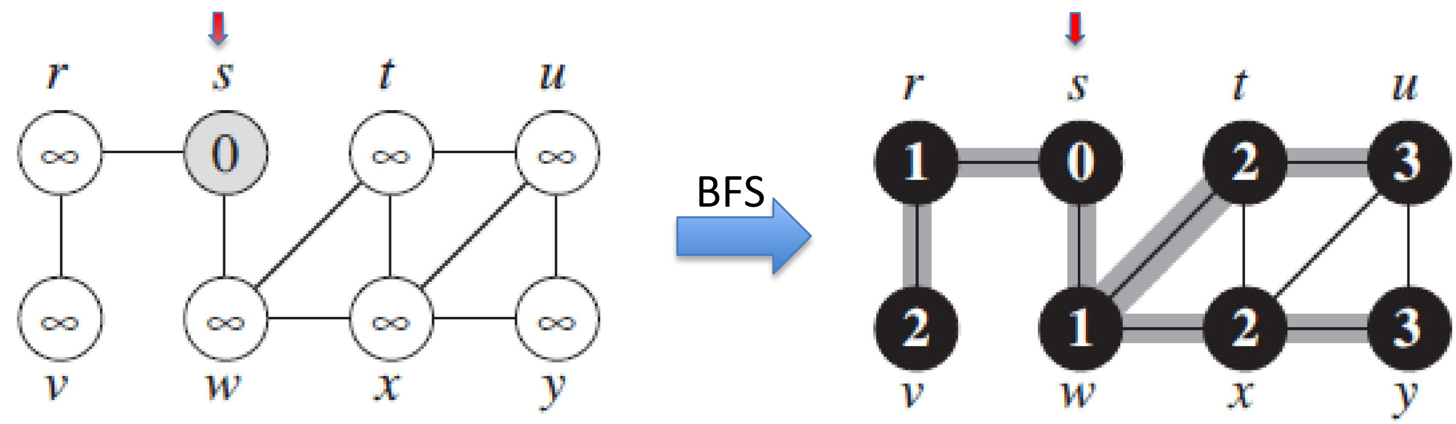


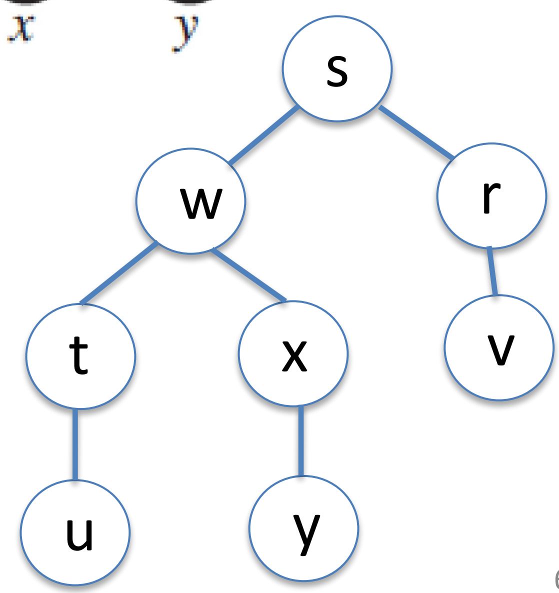
In-order traversal of binary search tree gives sorted list of objects



Breadth-First Search

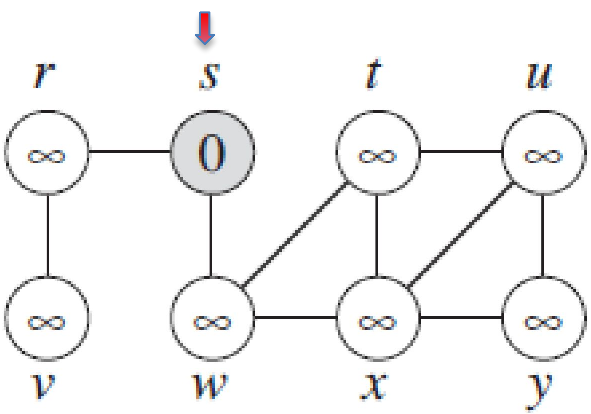
- Breadth-First Search (BFS)
  - computes a path from s to each reachable vertex
  - produces a “breadth-first tree” of all reachable vertices with starting vertex, s, as root
  - Useful in computing minimum-spanning tree & Dijkstra’s single-source shortest-paths

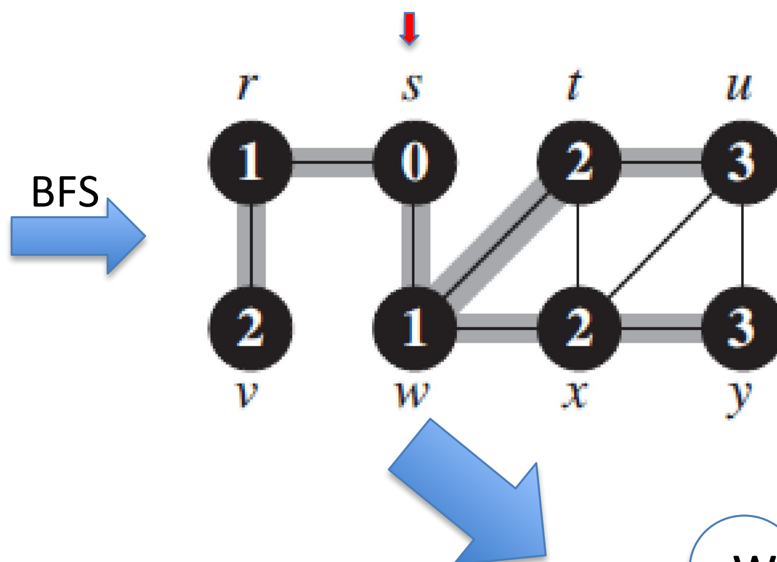


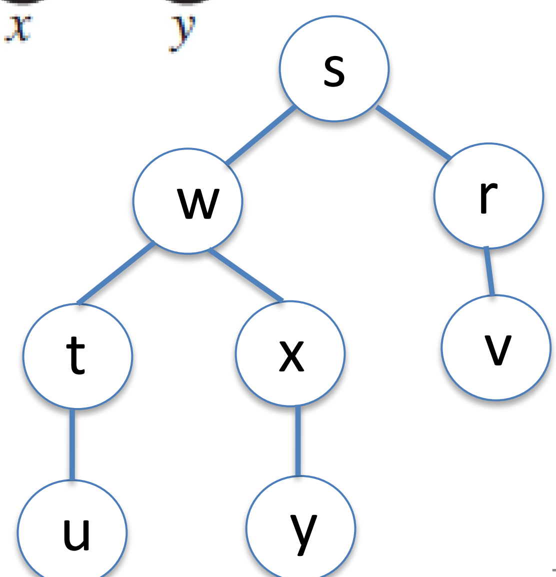


Breadth-First Search

- Breadth-First Search (BFS)
  - computes a path from s to each reachable vertex
  - produces a “breadth-first tree” of all reachable vertices with starting vertex, s, as root
  - Useful in computing minimum-spanning tree & Dijkstra’s single-source shortest-paths

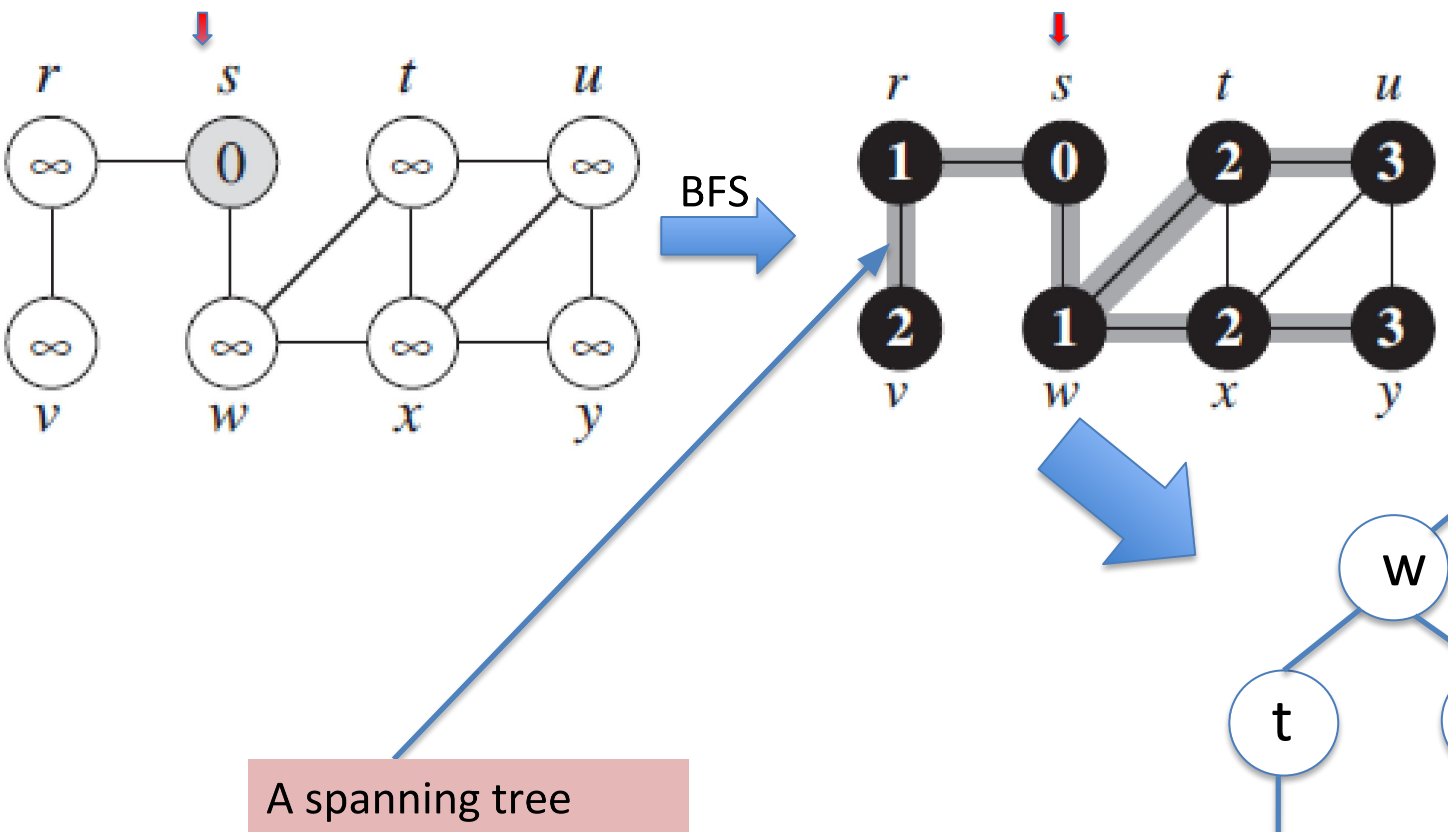


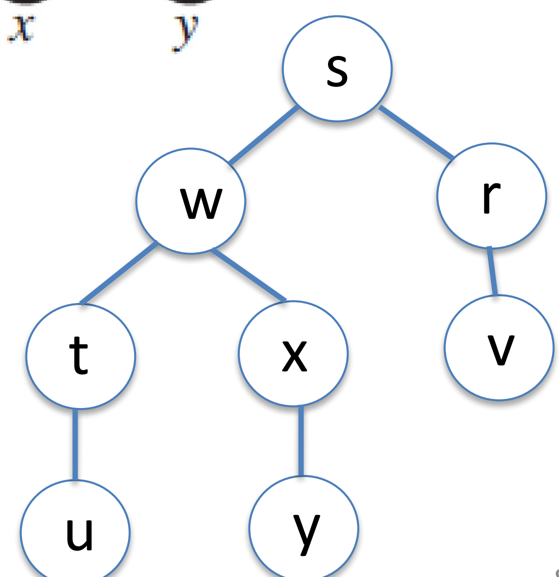




Breadth-First Search

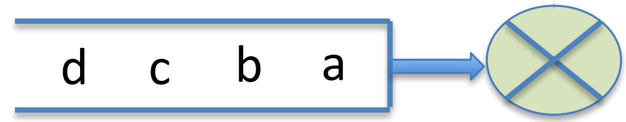
- Breadth-First Search (BFS)
  - computes a path from s to each reachable vertex
  - produces a "breadth-first tree" of all reachable vertices with starting vertex, s, as root
  - Useful in computing minimum-spanning tree & Dijkstra's single-source shortest-paths



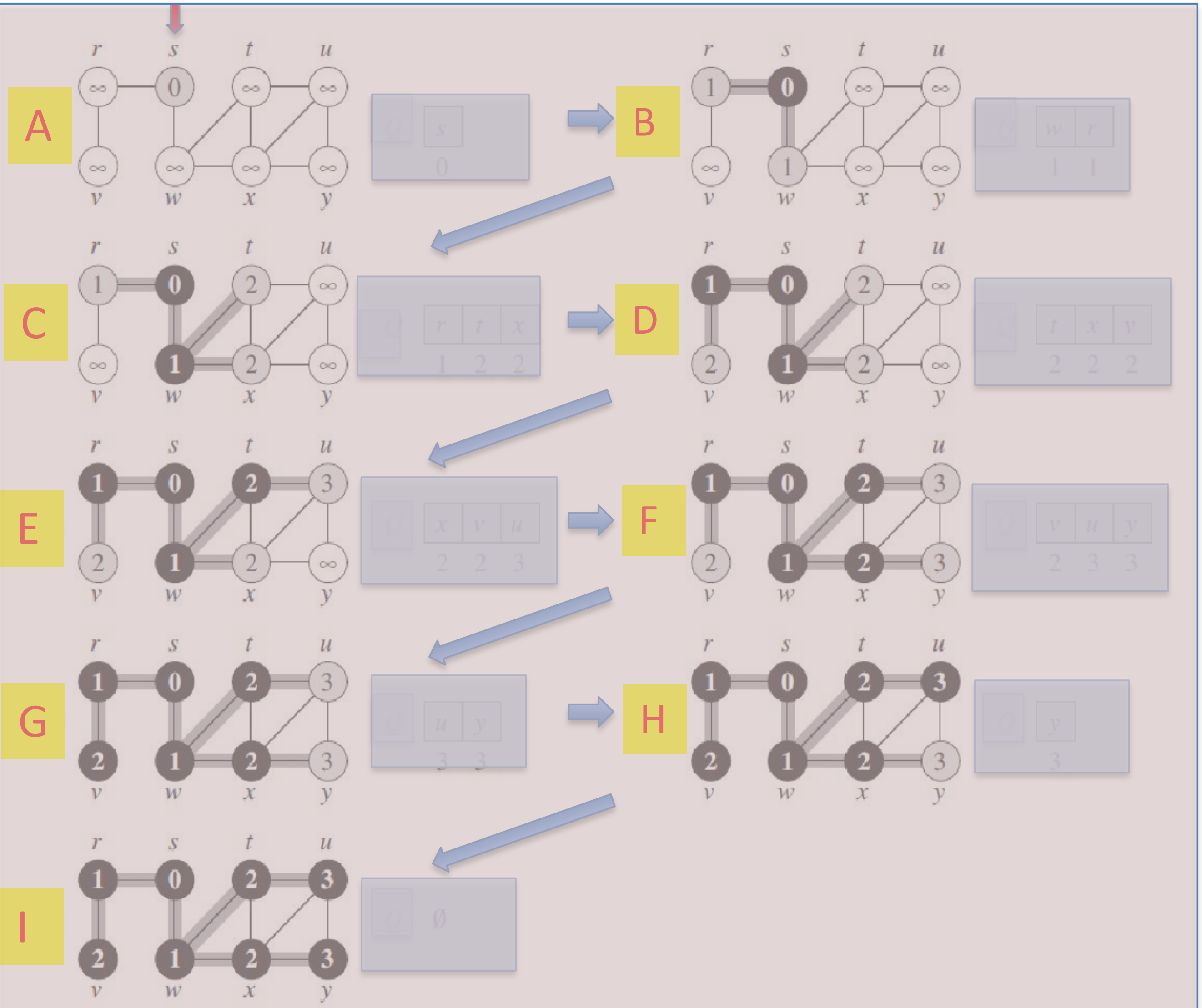


Breadth-First Search

- The need for a Queue, Q

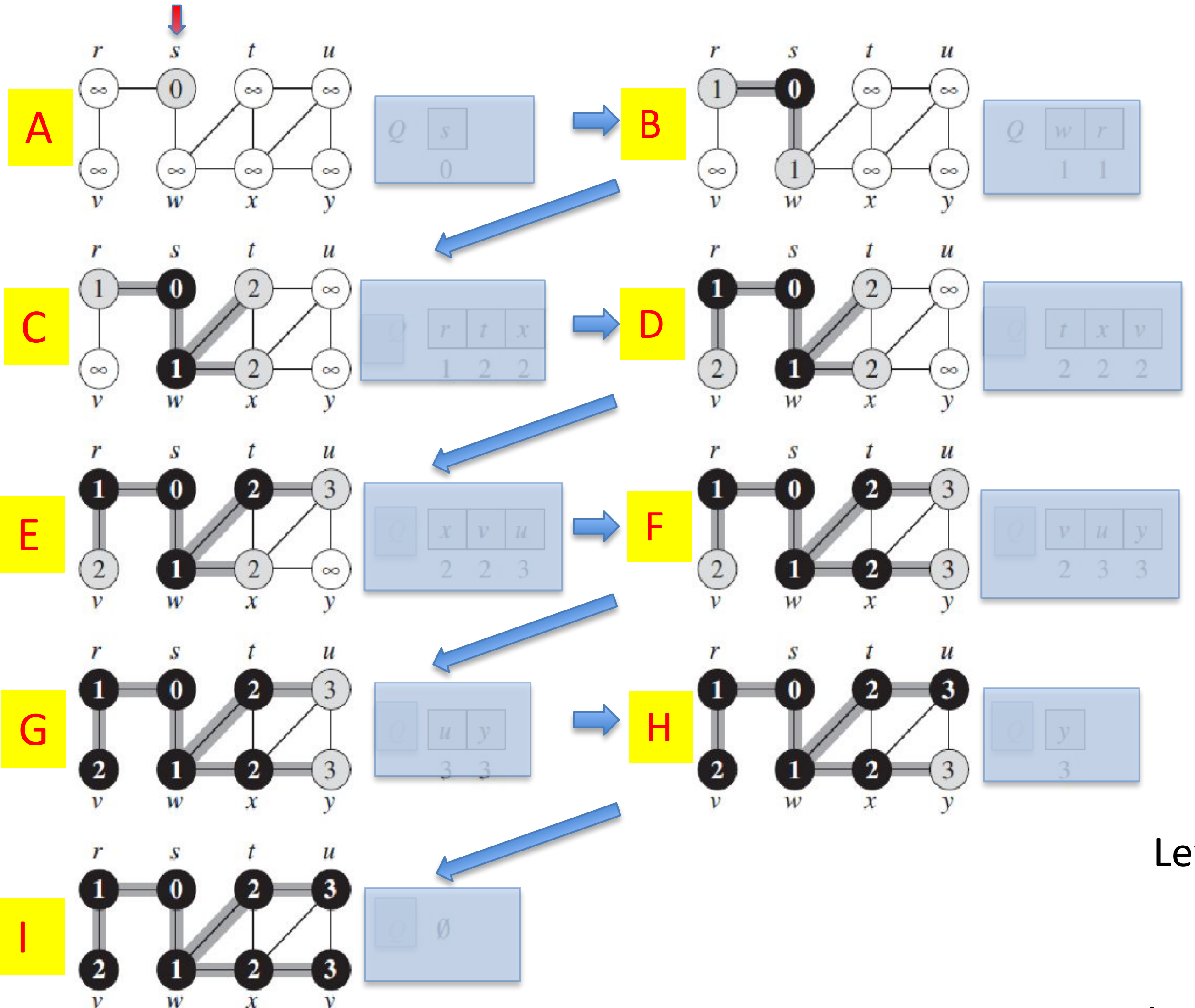


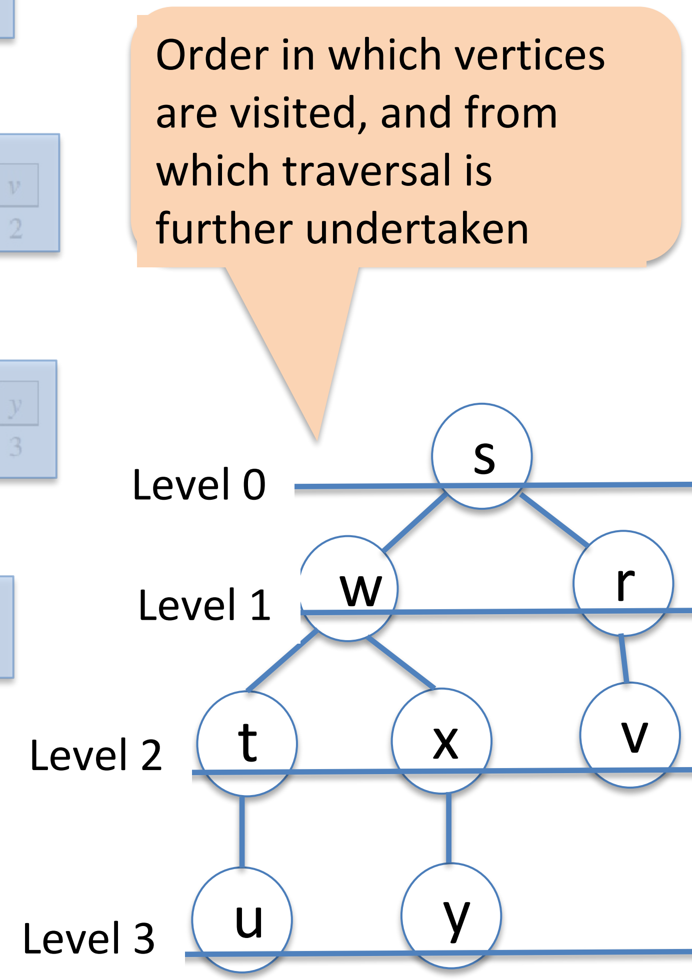
queue processor



Breadth-First Search

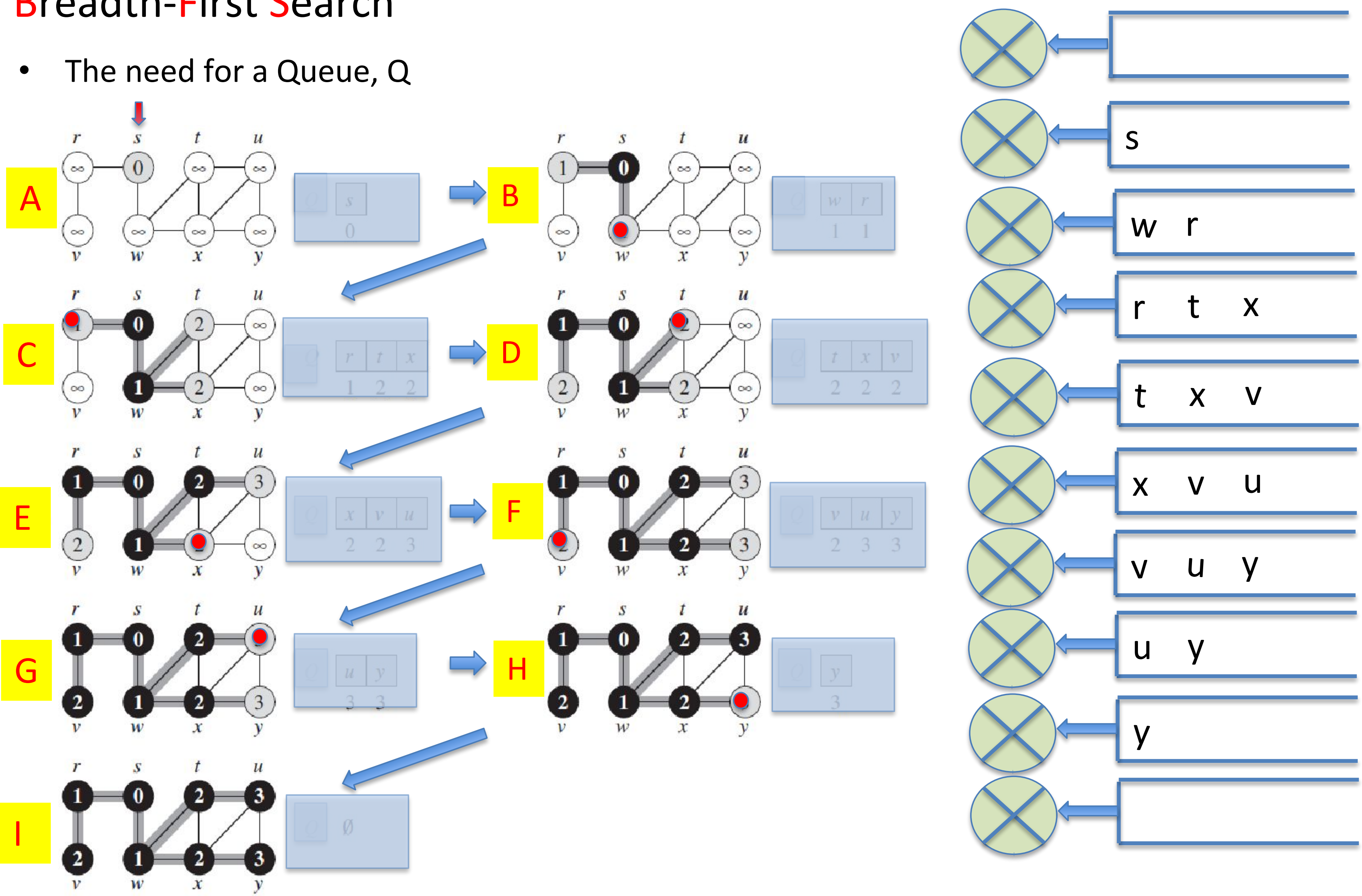
- The need for a Queue, Q





Breadth-First Search

- The need for a Queue, Q



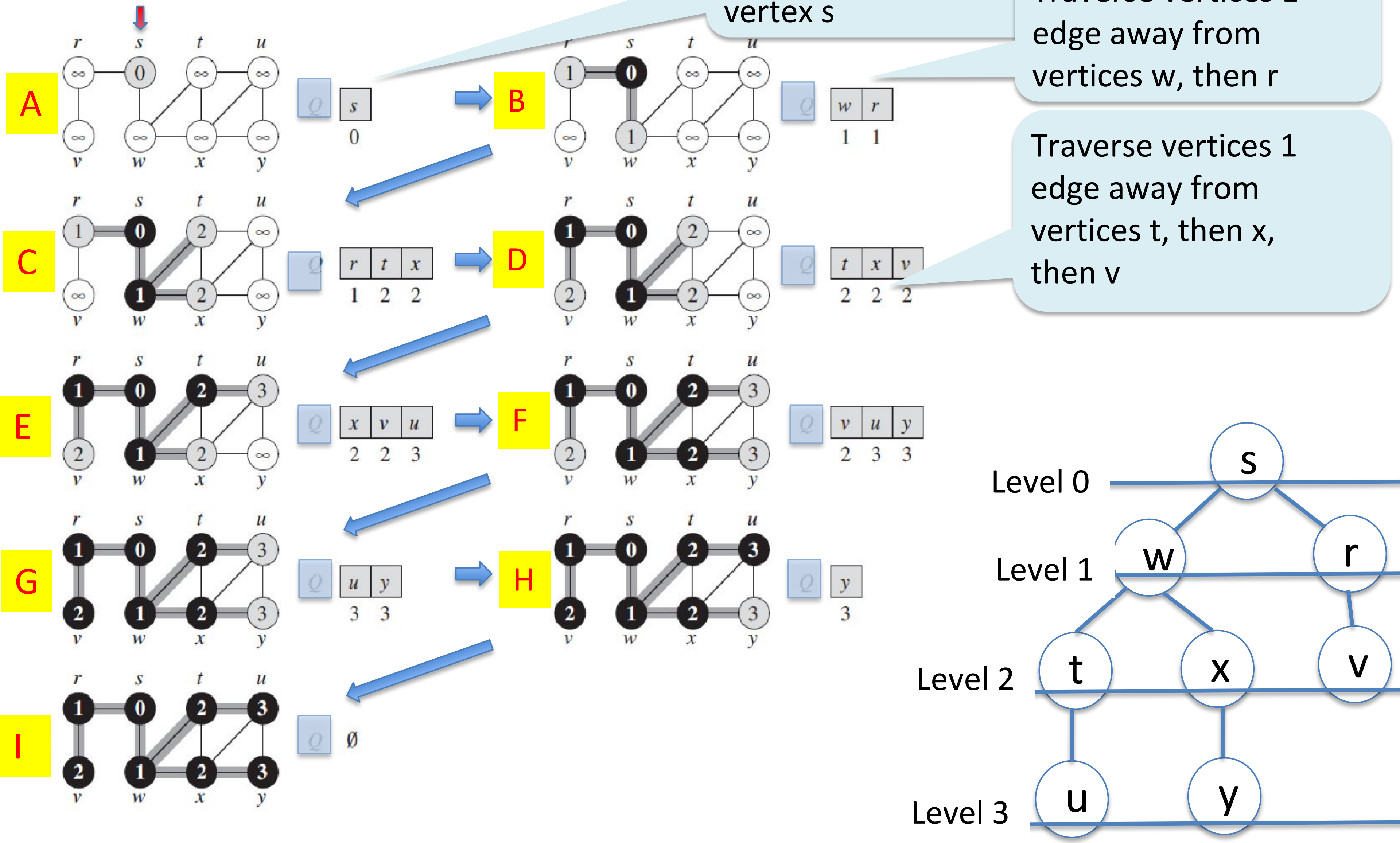
Breadth-First Search

- The need for a Queue, Q

Traverse vertices 1 edge away from vertex s

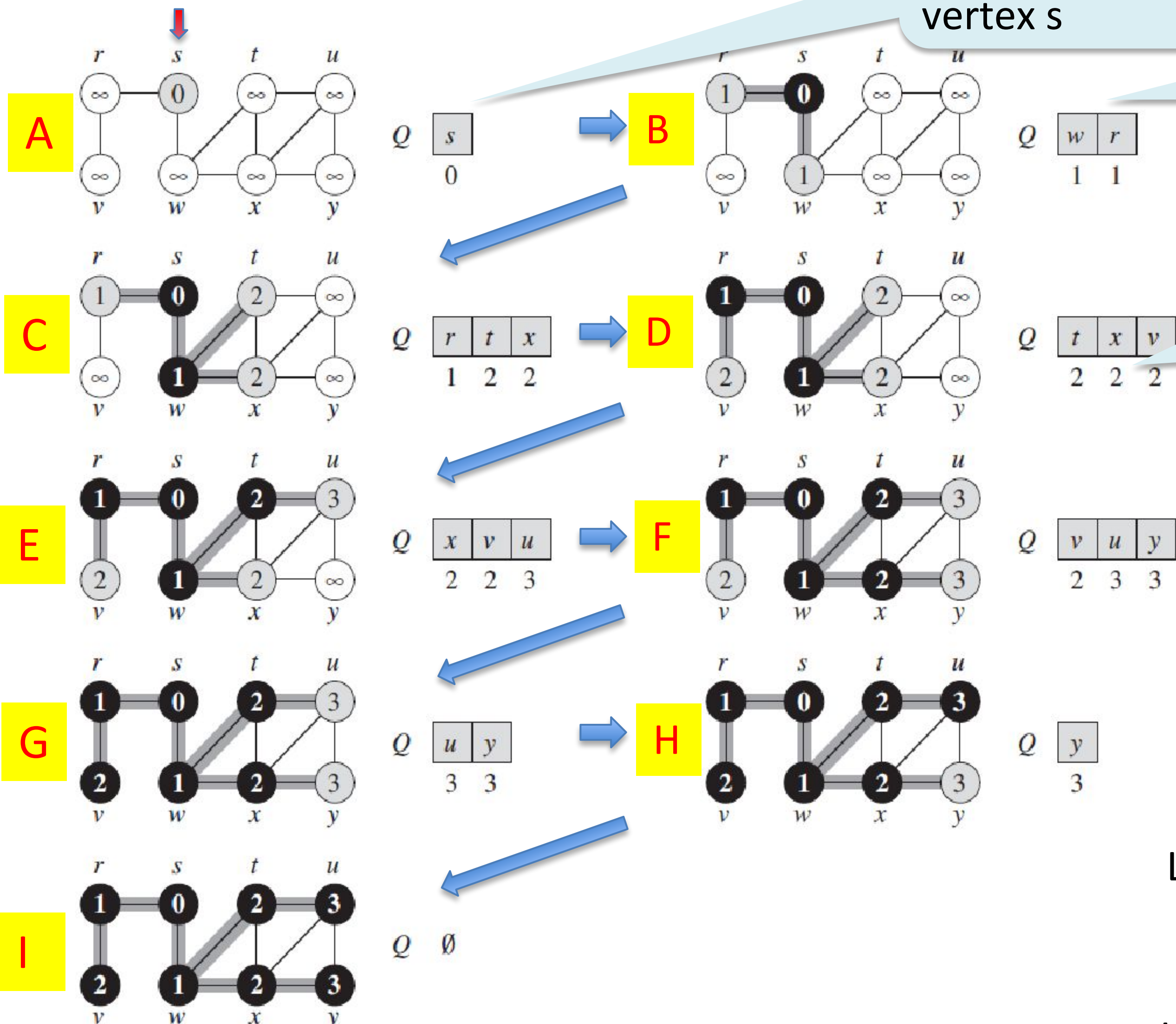
Traverse vertices 1 edge away from vertices w, then r

Traverse vertices 1 edge away from vertices t, then x, then v



Breadth-First Search

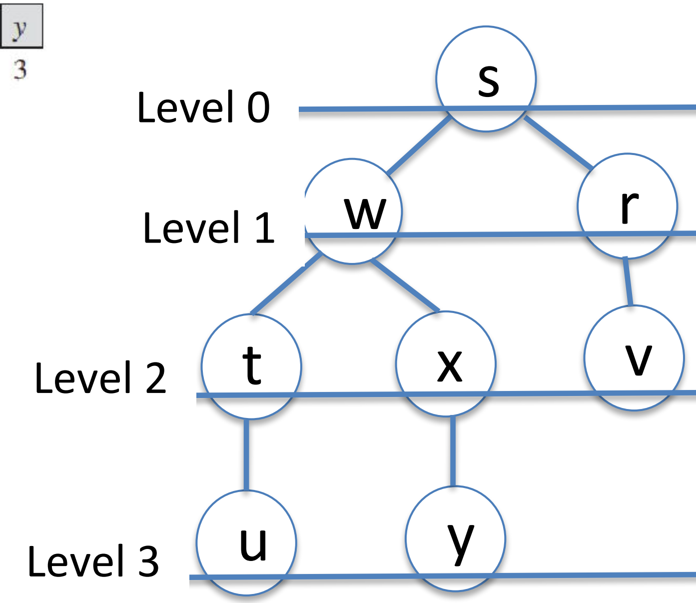
- The need for a Queue, Q



Traverse vertices 1 edge away from vertex s

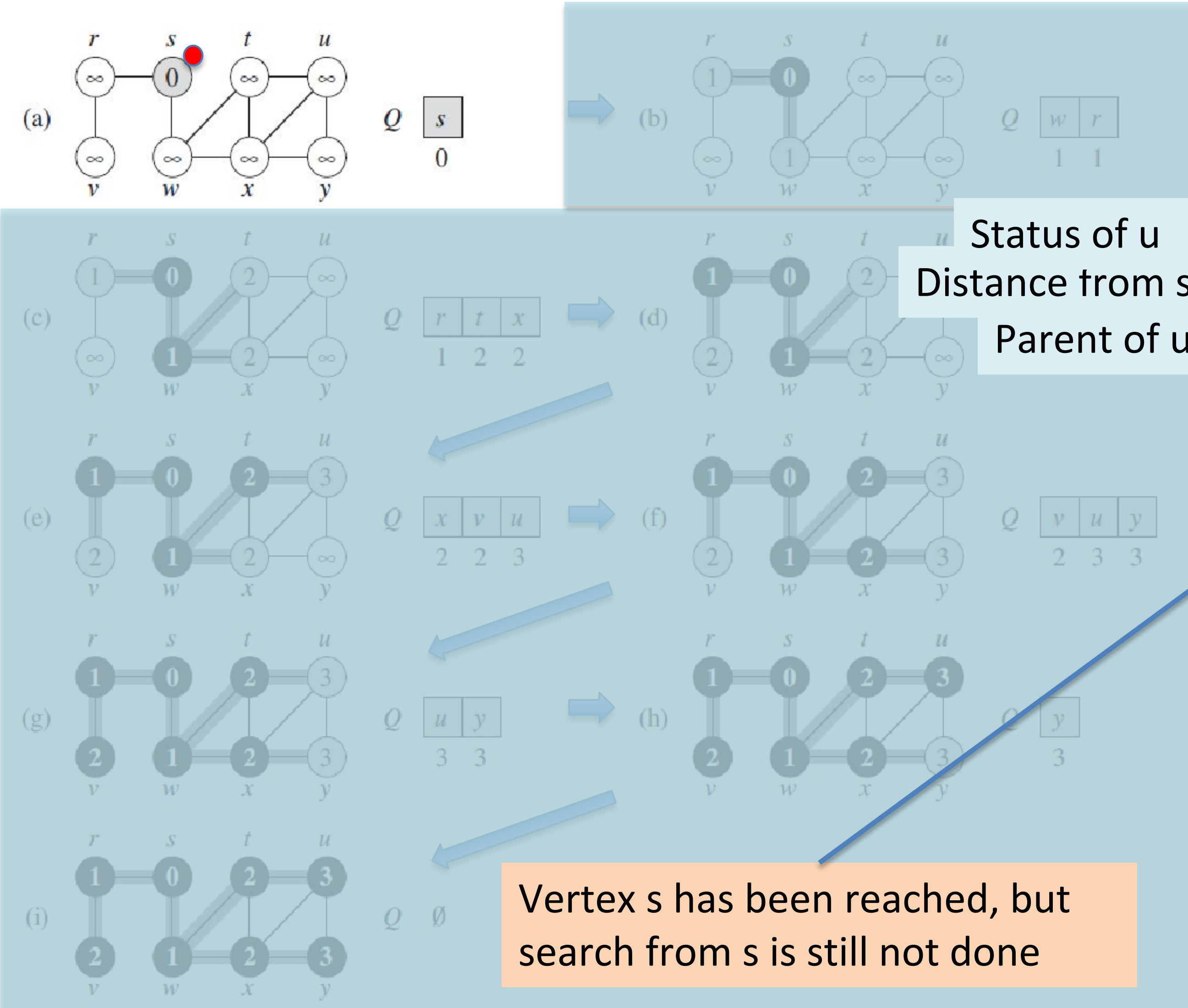
Traverse vertices 1 edge away from vertices w, then r

Traverse vertices 1 edge away from vertices t, then x, then v



Breadth-First Search

- Breadth-First Search (BFS)



- Identify the starting vertex, s
- Initialize:

for each vertex $u \in G.V - \{s\}$

Status of u → $u.color = \text{WHITE}$

Distance from s → $u.d = \infty$

Parent of u → $u.\pi = \text{NIL}$

$s.color = \text{GRAY}$

$s.d = 0$

$s.\pi = \text{NIL}$

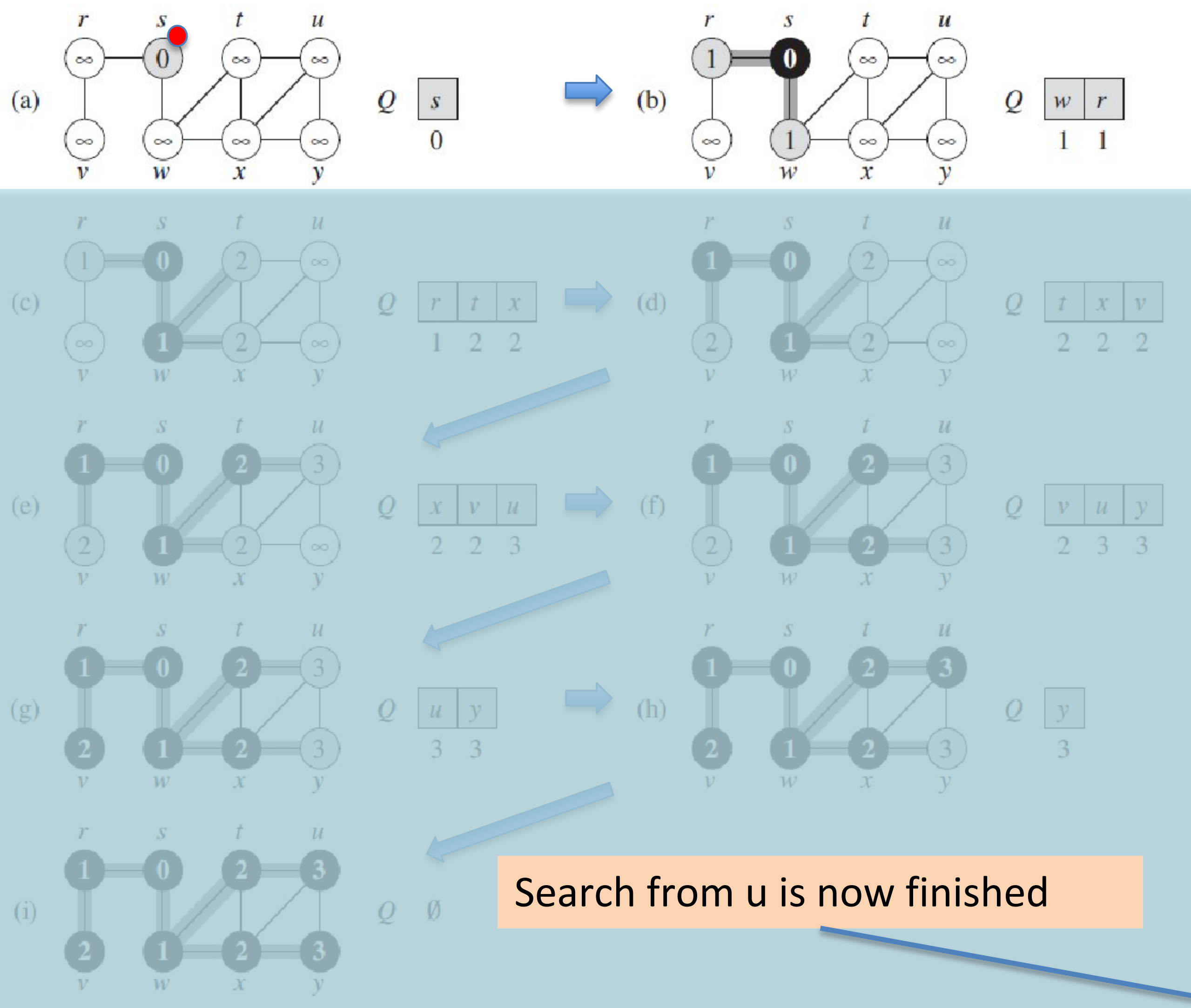
$Q = \emptyset$

$\text{ENQUEUE}(Q, s)$

Vertex s has been reached, but search from s is still not done

Breadth-First Search

- Breadth-First Search (BFS)



- Identify the starting vertex, s
- Initialize:

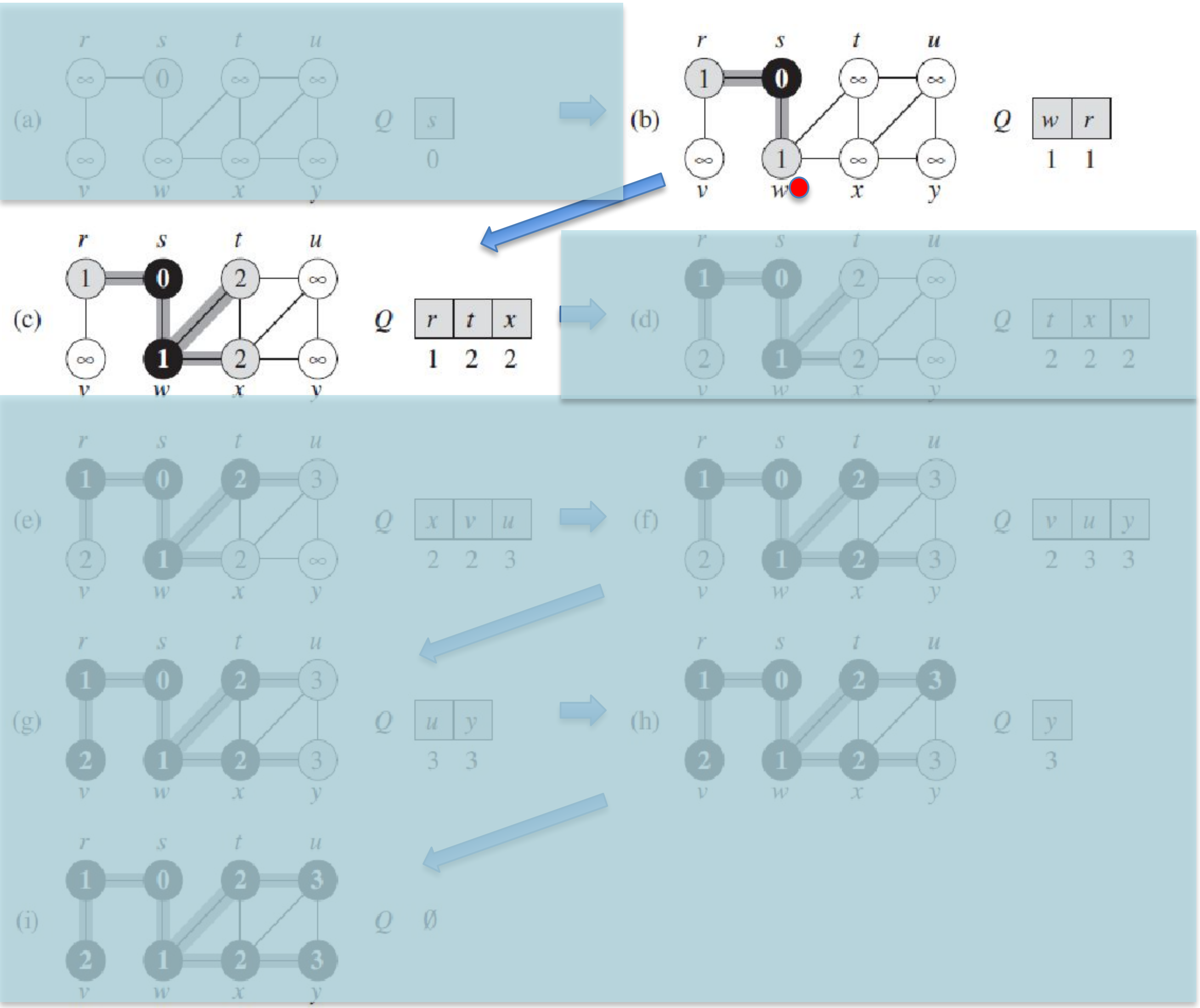
for each vertex $u \in G.V - \{s\}$
$u.color = \text{WHITE}$
$u.d = \infty$
$u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$
$Q = \emptyset$
$\text{ENQUEUE}(Q, s)$

- Then:

$u = \text{DEQUEUE}(Q)$
for each $v \in G.Adj[u]$
if $v.color == \text{WHITE}$
$v.color = \text{GRAY}$
$v.d = u.d + 1$
$v.\pi = u$
$\text{ENQUEUE}(Q, v)$
$u.color = \text{BLACK}$

Breadth-First Search

- Breadth-First Search (BFS)



- Identify the starting vertex, s
- Initialize:

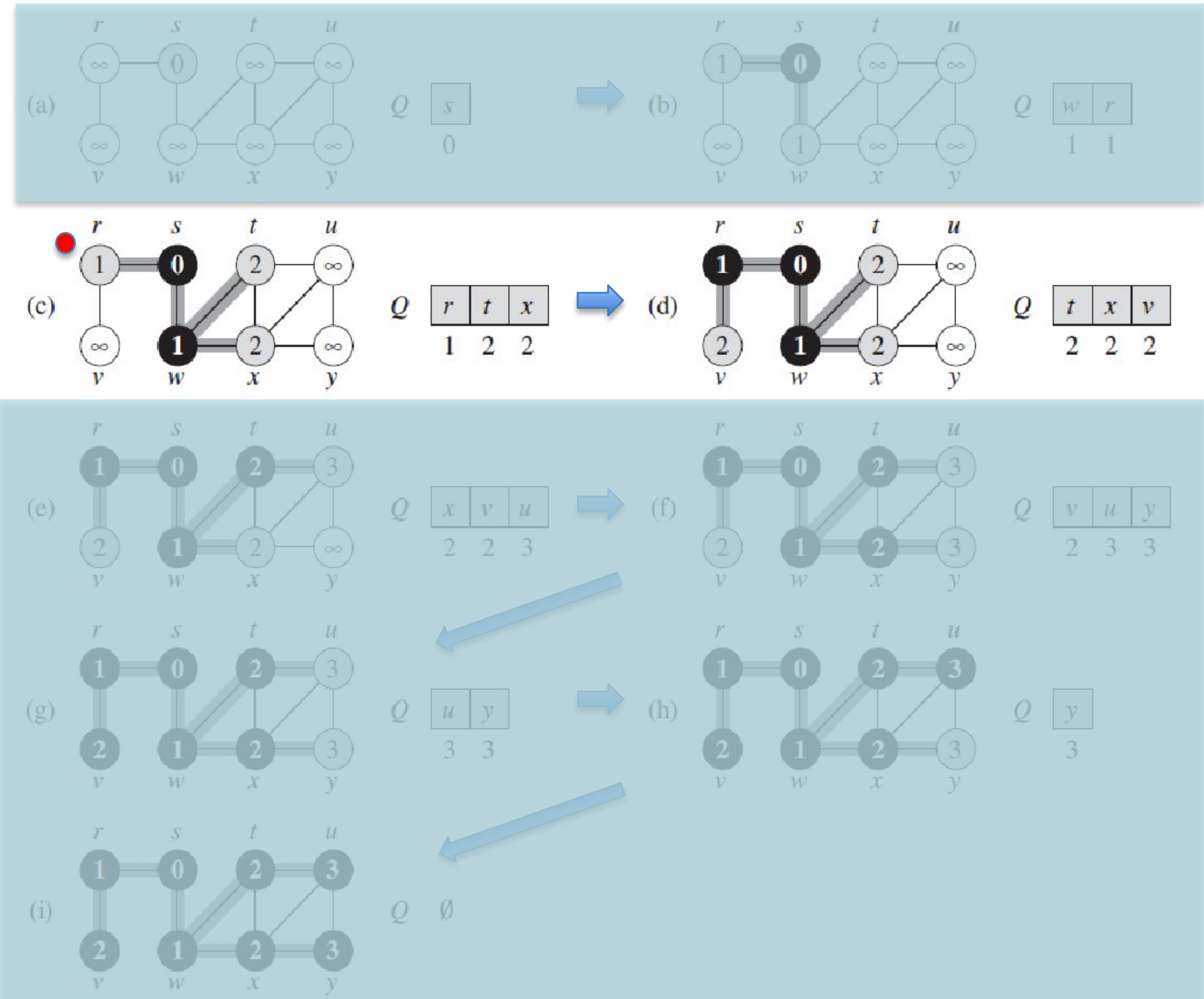
for each vertex $u \in G.V - \{s\}$
    $u.color = \text{WHITE}$
    $u.d = \infty$
    $u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$
$Q = \emptyset$
$\text{ENQUEUE}(Q, s)$

- Then:

while $Q \neq \emptyset$
    $u = \text{DEQUEUE}(Q)$
    for each $v \in G.Adj[u]$
        if $v.color == \text{WHITE}$
            $v.color = \text{GRAY}$
            $v.d = u.d + 1$
            $v.\pi = u$
            $\text{ENQUEUE}(Q, v)$
    $u.color = \text{BLACK}$

Breadth-First Search

- Breadth-First Search (BFS)



- Identify the starting vertex, s
- Initialize:

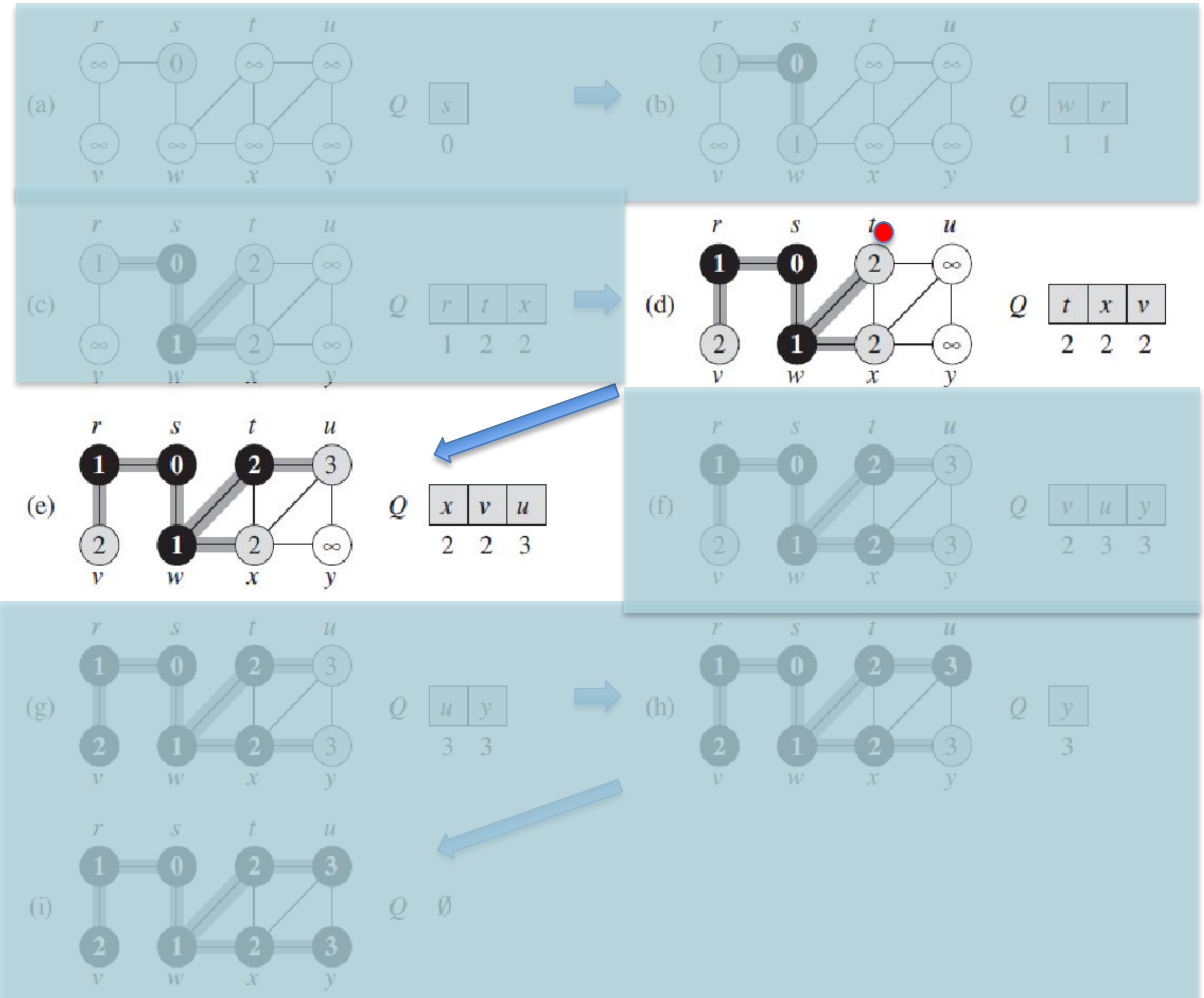
for each vertex $u \in G.V - \{s\}$
  $u.color = \text{WHITE}$
  $u.d = \infty$
  $u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$
$Q = \emptyset$
$\text{ENQUEUE}(Q, s)$

- Then:

while $Q \neq \emptyset$
  $u = \text{DEQUEUE}(Q)$
  for each $v \in G.Adj[u]$
    if $v.color == \text{WHITE}$
      $v.color = \text{GRAY}$
      $v.d = u.d + 1$
      $v.\pi = u$
      $\text{ENQUEUE}(Q, v)$
  $u.color = \text{BLACK}$

Breadth-First Search

- Breadth-First Search (BFS)



- Identify the starting vertex, s
- Initialize:

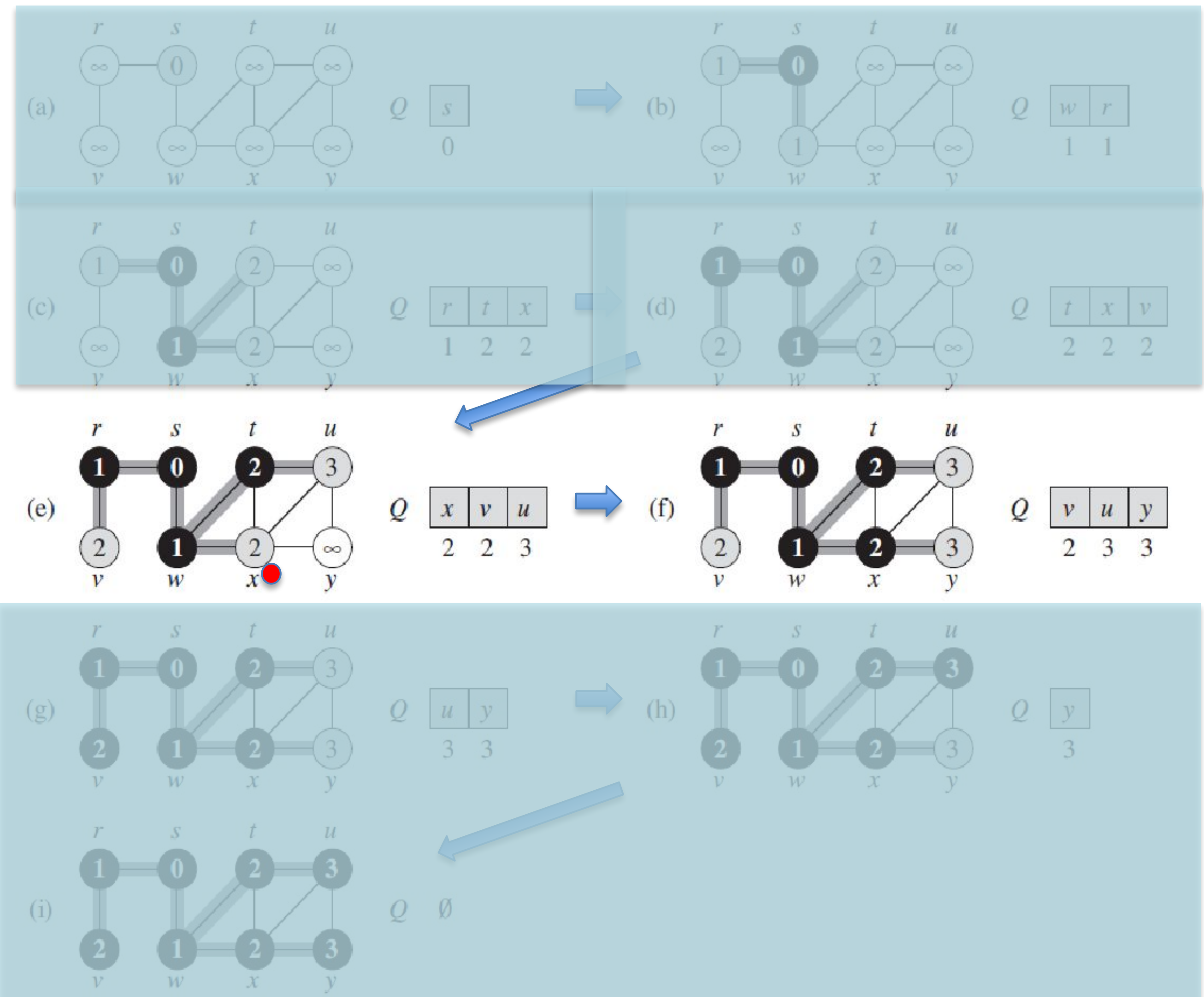
```
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
```

- Then:

```
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

- Breadth-First Search (BFS)



- Identify the starting vertex, s
- Initialize:

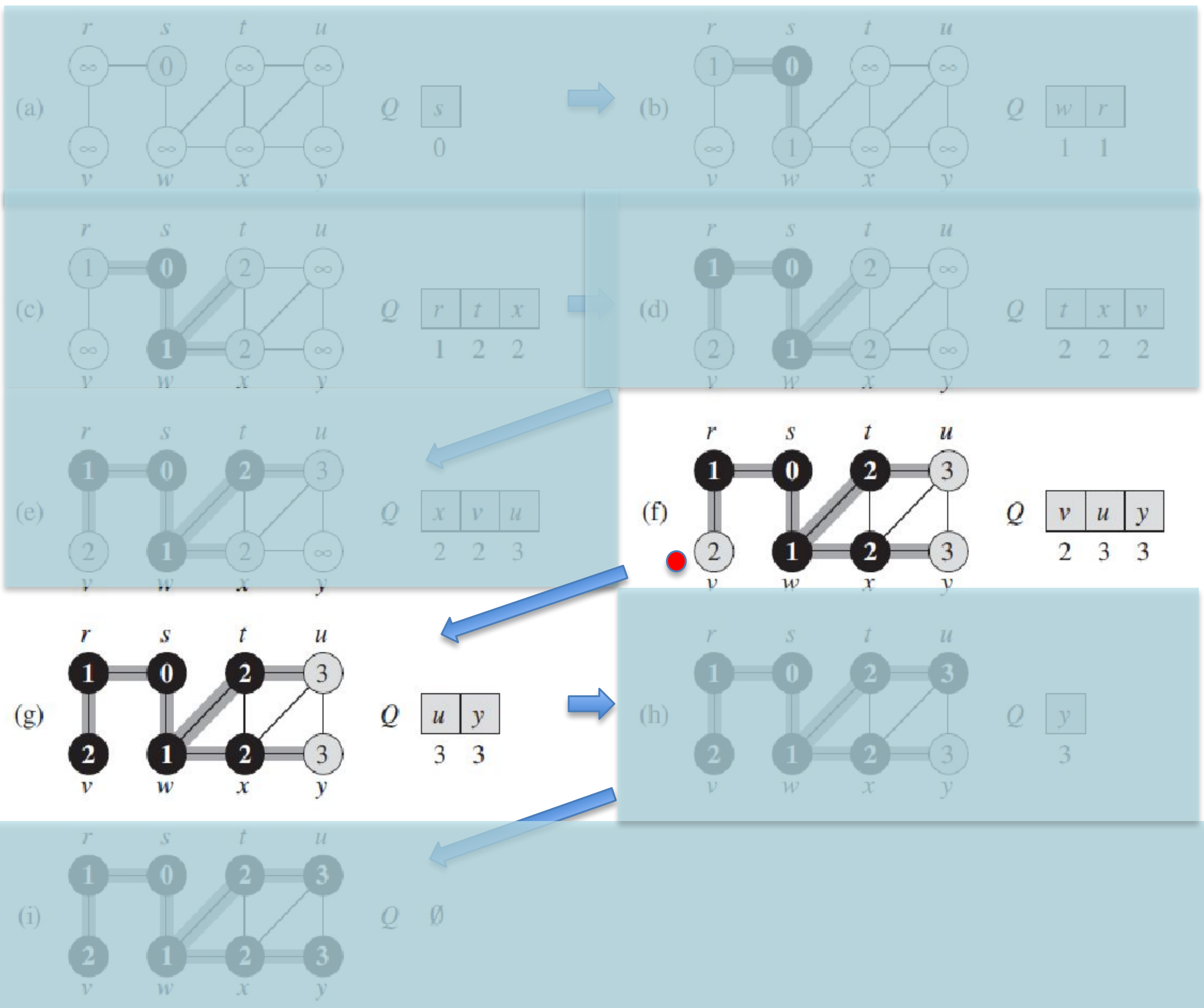
for each vertex $u \in G.V - \{s\}$
    $u.color = \text{WHITE}$
    $u.d = \infty$
    $u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$
$Q = \emptyset$
$\text{ENQUEUE}(Q, s)$

- Then:

while $Q \neq \emptyset$
    $u = \text{DEQUEUE}(Q)$
    for each $v \in G.Adj[u]$
        if $v.color == \text{WHITE}$
            $v.color = \text{GRAY}$
            $v.d = u.d + 1$
            $v.\pi = u$
            $\text{ENQUEUE}(Q, v)$
    $u.color = \text{BLACK}$

Breadth-First Search

- Breadth-First Search (BFS)



- Identify the starting vertex, s
- Initialize:

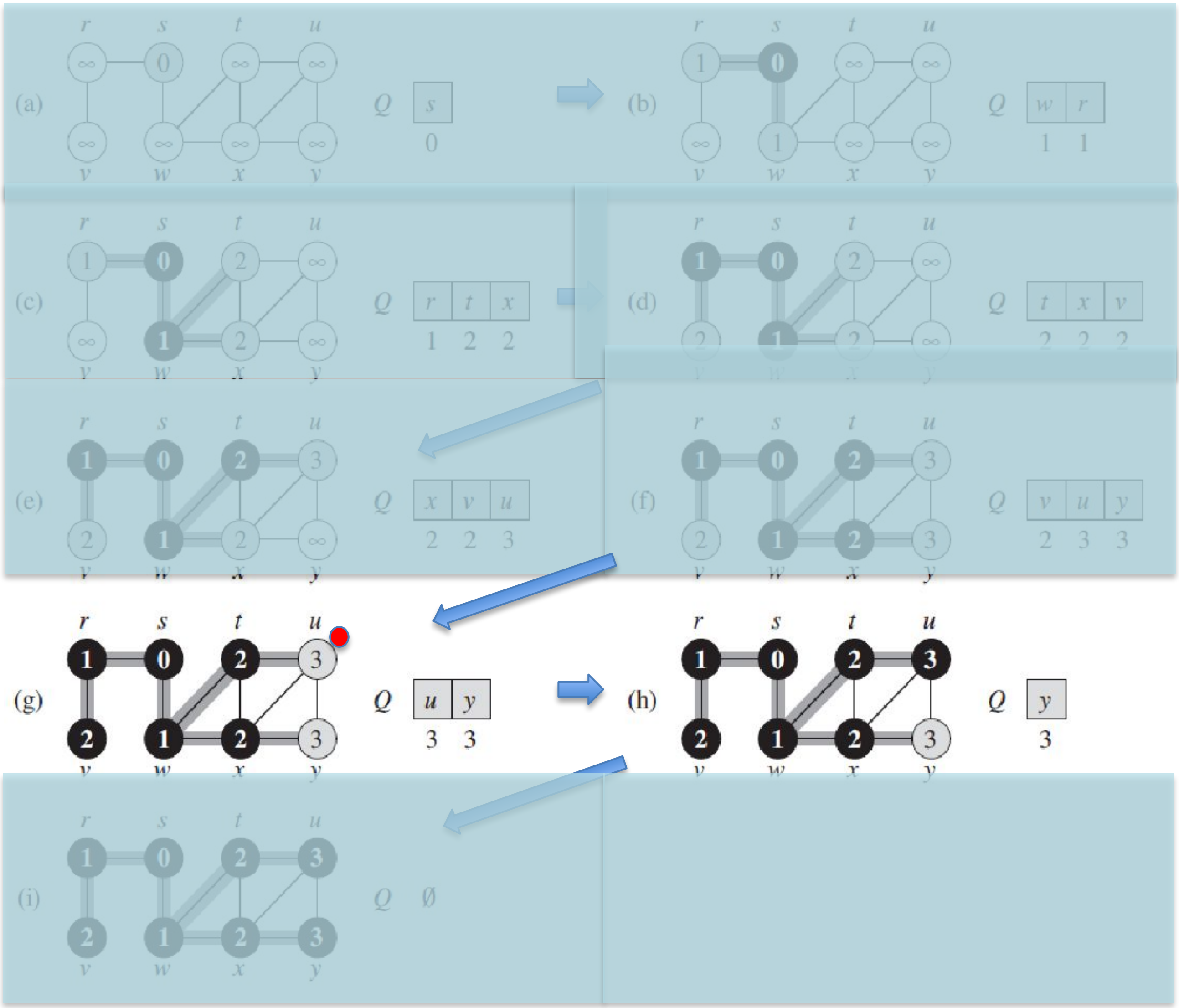
for each vertex $u \in G.V - \{s\}$
  $u.color = \text{WHITE}$
  $u.d = \infty$
  $u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$
$Q = \emptyset$
$\text{ENQUEUE}(Q, s)$

- Then:

while $Q \neq \emptyset$
  $u = \text{DEQUEUE}(Q)$
  for each $v \in G.Adj[u]$
    if $v.color == \text{WHITE}$
      $v.color = \text{GRAY}$
      $v.d = u.d + 1$
      $v.\pi = u$
      $\text{ENQUEUE}(Q, v)$
  $u.color = \text{BLACK}$

Breadth-First Search

- Breadth-First Search (BFS)



- Identify the starting vertex, s
- Initialize:

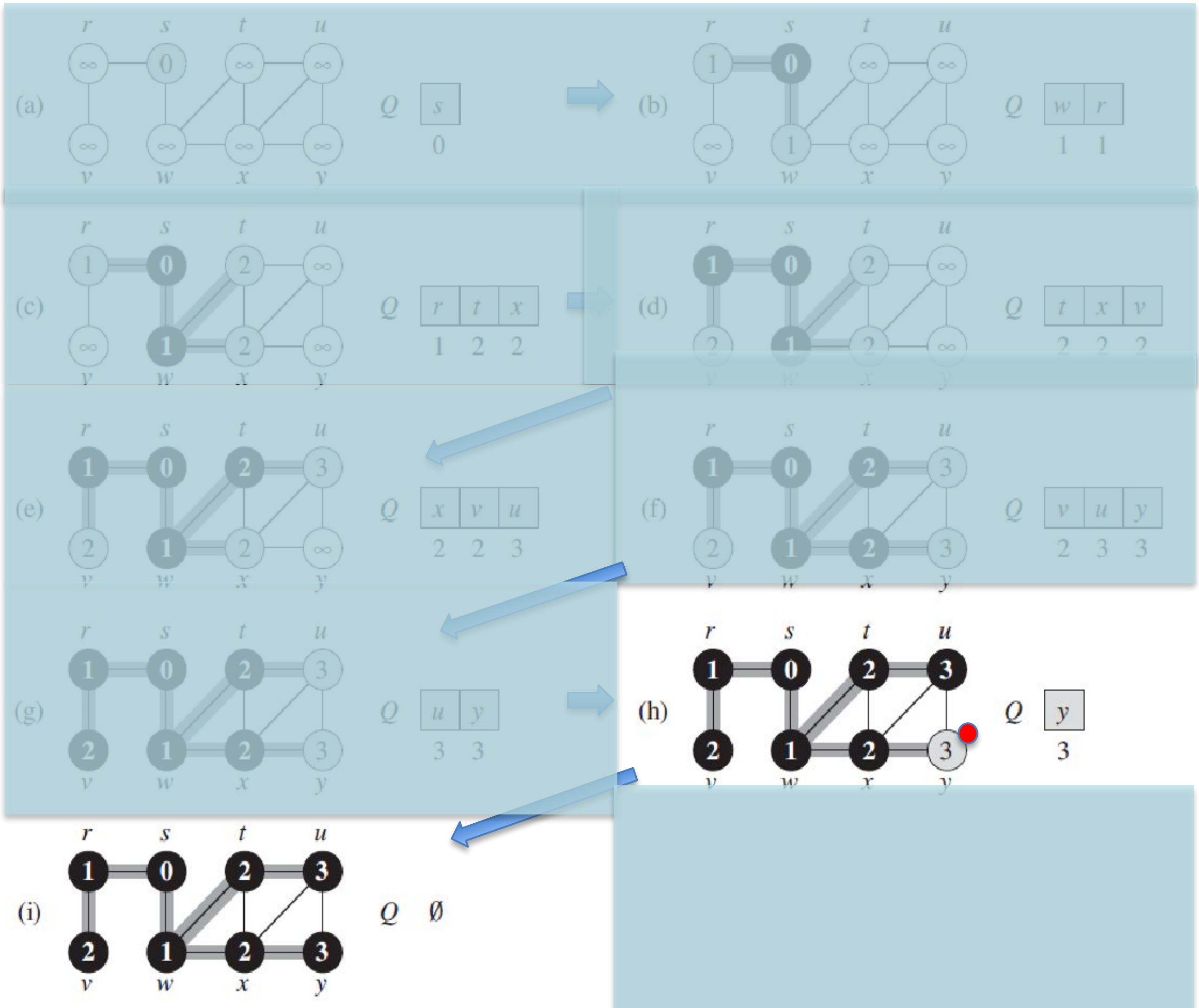
for each vertex $u \in G.V - \{s\}$
    $u.color = \text{WHITE}$
    $u.d = \infty$
    $u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$
$Q = \emptyset$
$\text{ENQUEUE}(Q, s)$

- Then:

while $Q \neq \emptyset$
    $u = \text{DEQUEUE}(Q)$
    for each $v \in G.Adj[u]$
        if $v.color == \text{WHITE}$
            $v.color = \text{GRAY}$
            $v.d = u.d + 1$
            $v.\pi = u$
            $\text{ENQUEUE}(Q, v)$
    $u.color = \text{BLACK}$

Breadth-First Search

- Breadth-First Search (BFS)



- Identify the starting vertex, s
- Initialize:

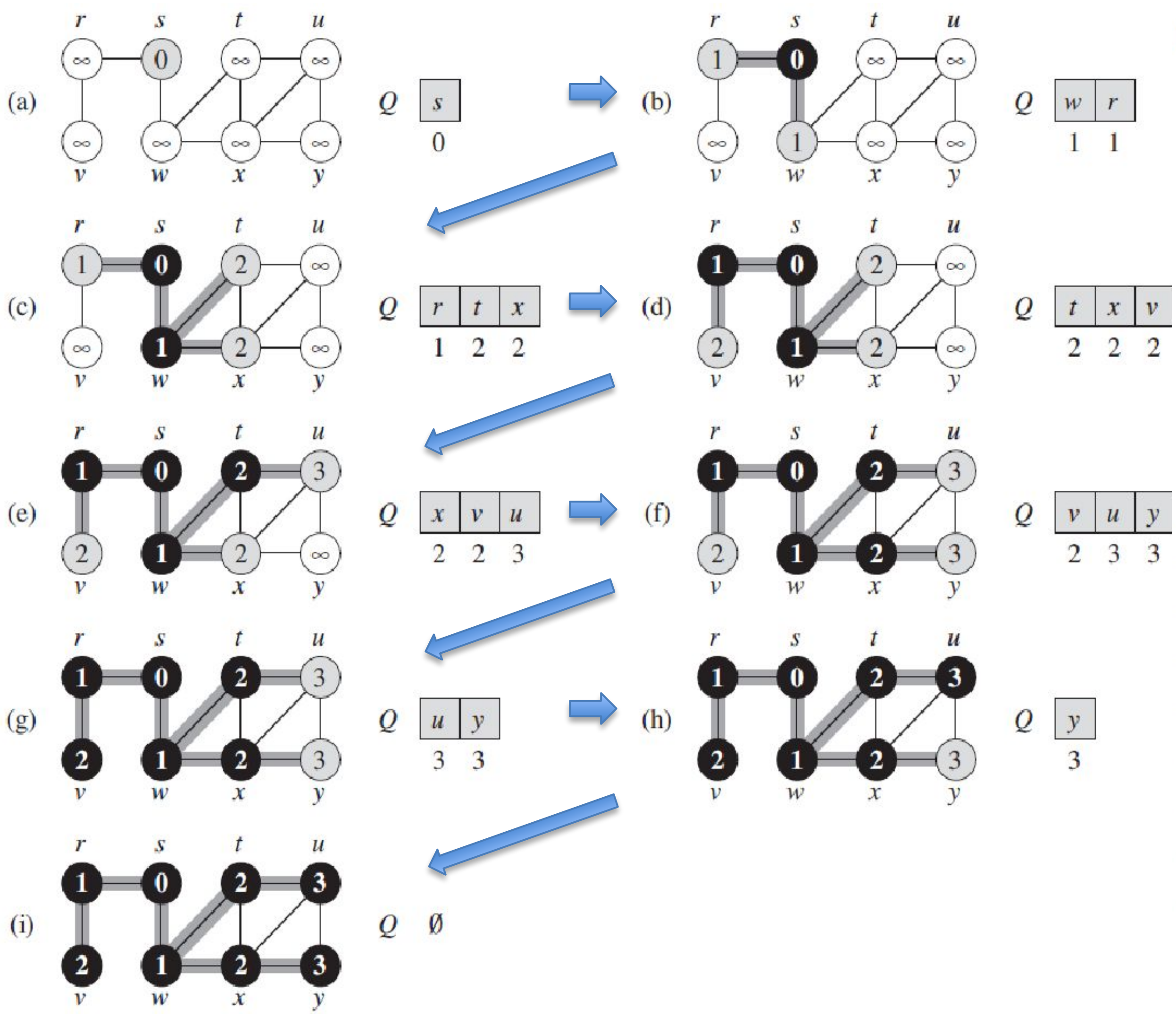
for each vertex $u \in G.V - \{s\}$
$u.color = \text{WHITE}$
$u.d = \infty$
$u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$
$Q = \emptyset$
$\text{ENQUEUE}(Q, s)$

- Then:

while $Q \neq \emptyset$
$u = \text{DEQUEUE}(Q)$
for each $v \in G.Adj[u]$
if $v.color == \text{WHITE}$
$v.color = \text{GRAY}$
$v.d = u.d + 1$
$v.\pi = u$
$\text{ENQUEUE}(Q, v)$
$u.color = \text{BLACK}$

Breadth-First Search

- Breadth-First Search (BFS)

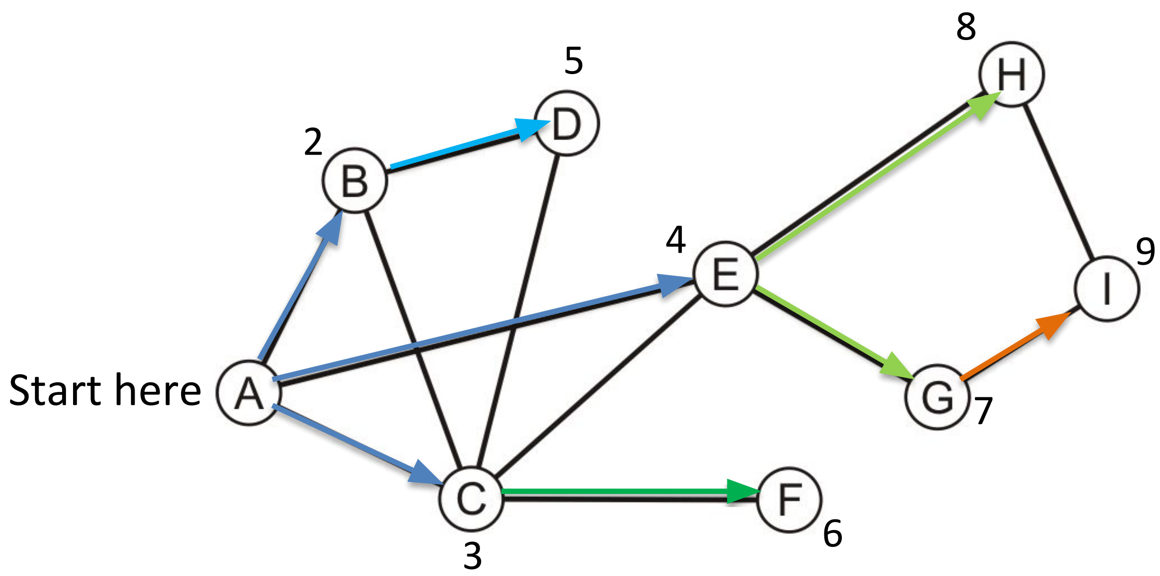


```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u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

BFS: another example

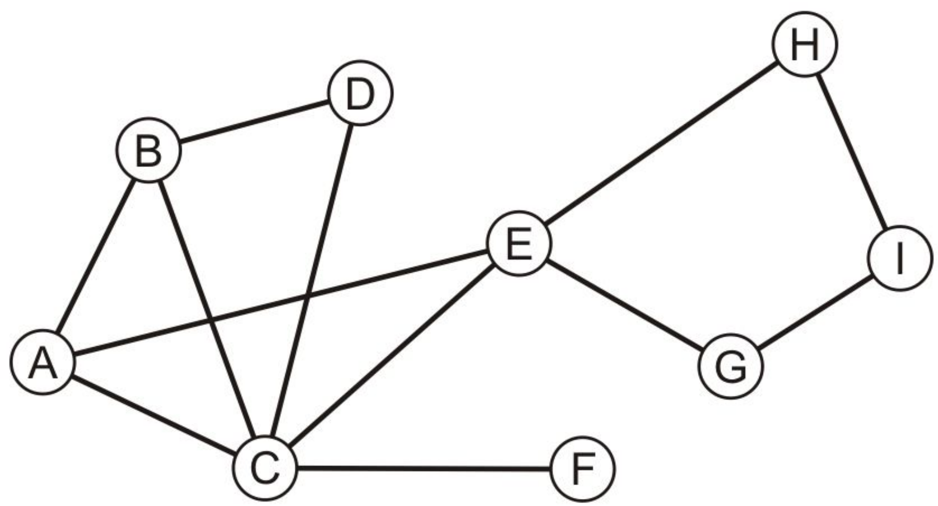
- Try your luck with this graph, starting the search from A
  - One sequence in which vertices are visited or processed is:

    A, B, C, E, D, F, G, H, I



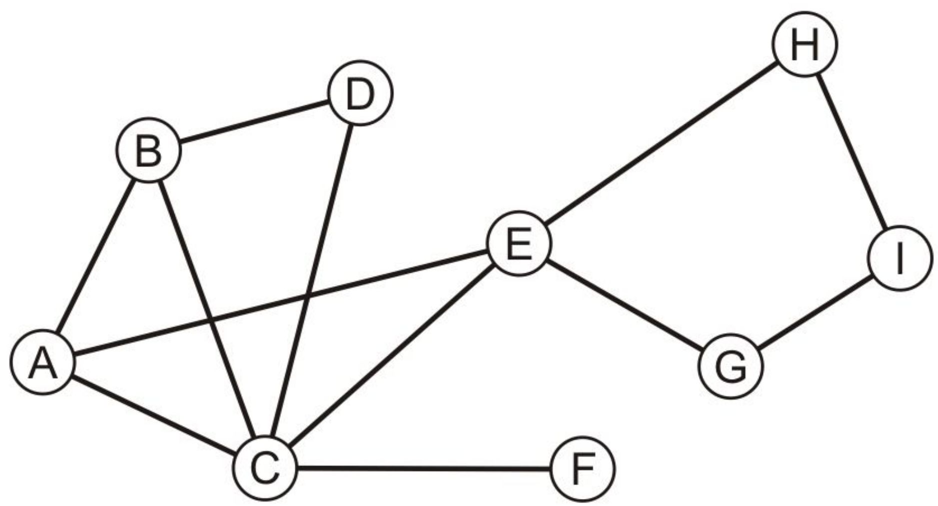
BFS: another example

- Try your luck with this graph, starting the search from A
  - One sequence in which vertices are visited or processed is:
    A, B, C, E, D, F, G, H, I
  - Could BFS ever visit vertices in a different sequence, such as A, B, C, D, E, G, I, H, F
  - Go through BFS to discover at least 2 other sequences in which vertices are visited



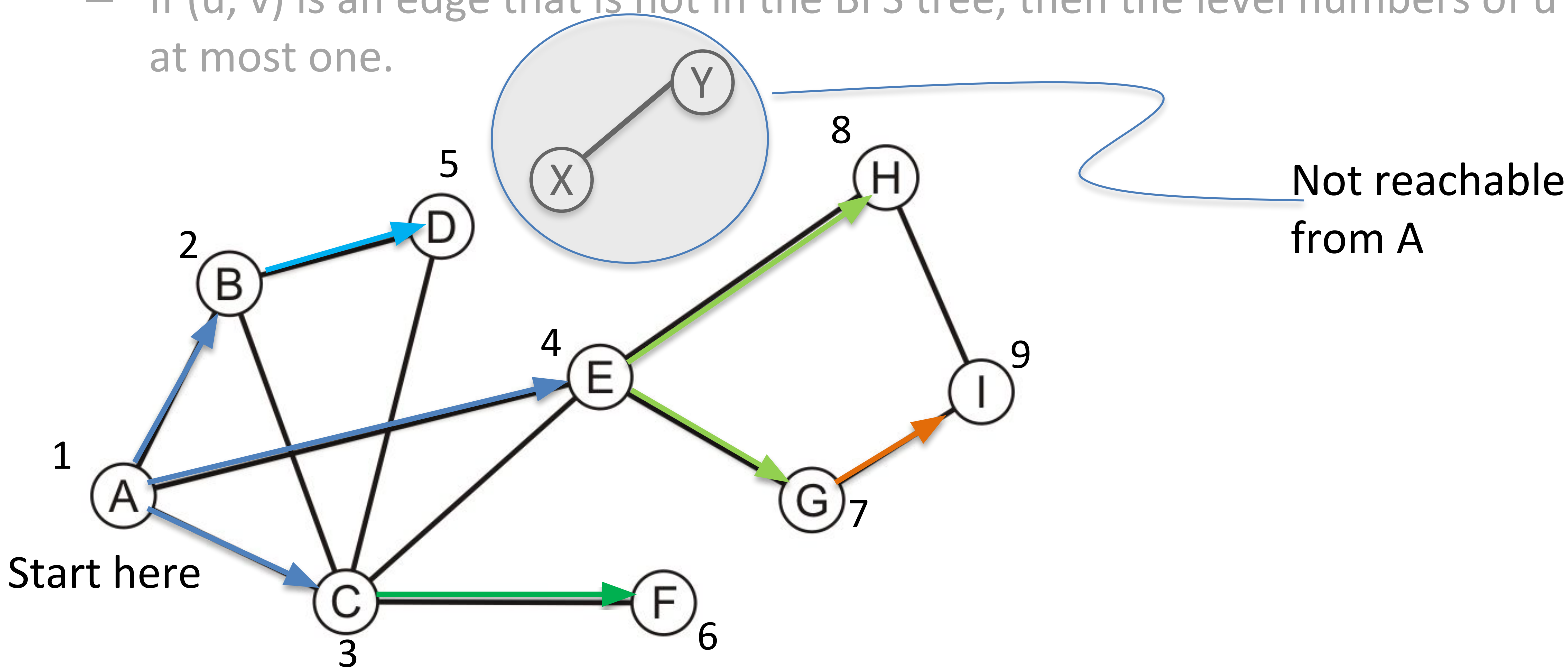
BFS: another example

- Try your luck with this graph, starting the search from A
  - One sequence in which vertices are visited or processed is:
    A, B, C, E, D, F, G, H, I
  - Could BFS ever visit vertices in a different sequence, such as A, B, C, D, E, G, I, H, F
  - Go through BFS to discover at least 2 other sequences in which vertices are visited



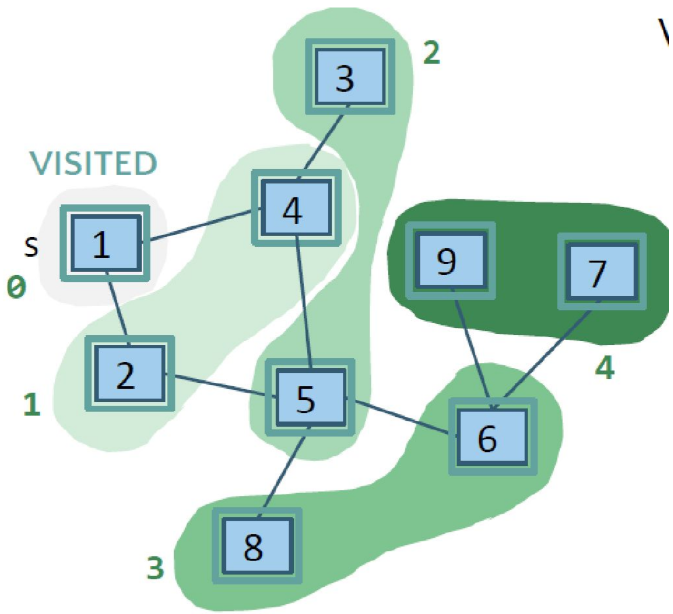
Properties of BFS

- Let G be an undirected graph on which BFS traversal starting at vertex s has been performed.
- Then
  - BFS traversal visits all vertices in the connected component that has vertex s.
  - Vertices are visited in order of “level” or how far they are from s
  - The discovery-edges form a spanning tree T. We call this “BFS tree” of the connected component of s
  - For each vertex v at level i, the path of the BFS tree T between s and v has i edges, and any other path of G between s and v has at least i edges.
  - If (u, v) is an edge that is not in the BFS tree, then the level numbers of u and v differ by at most one.



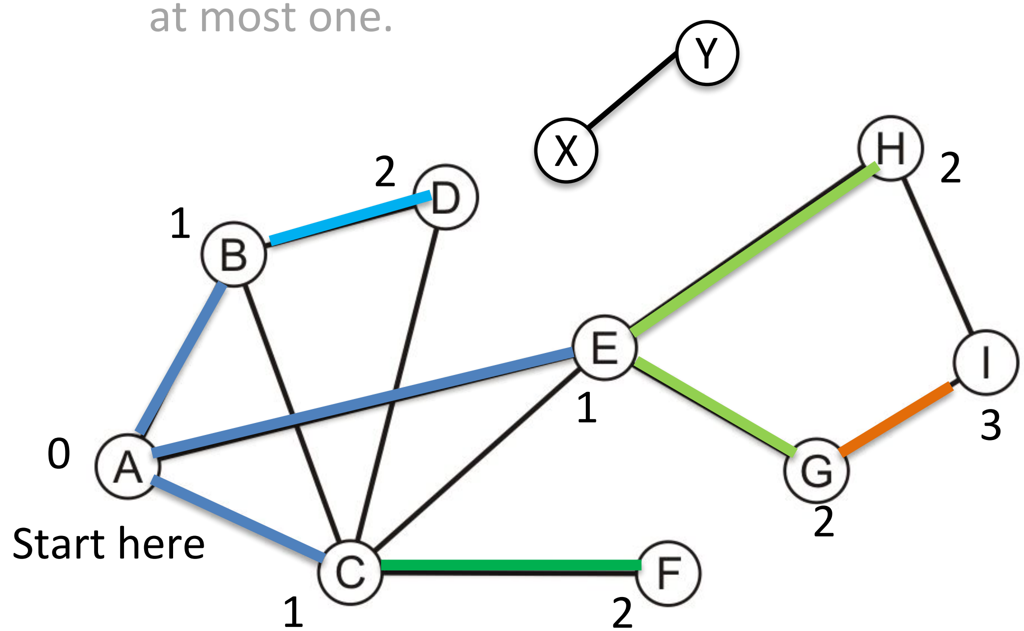
Properties of BFS

- Let G be an undirected graph on which BFS traversal starting at vertex s has been performed.
- Then
  - BFS traversal visits all vertices in the connected component that has vertex s.
  - Vertices are visited in order of "level" or how far they are from s
  - The discovery-edges form a spanning tree T. We call this "BFS tree" of the connected component of s
  - For each vertex v at level i, the path of the BFS tree T between s and v has i edges, and any other path of G between s and v has at least i edges.
  - If (u, v) is an edge that is not in the BFS tree, then the level numbers of u and v differ by at most one.



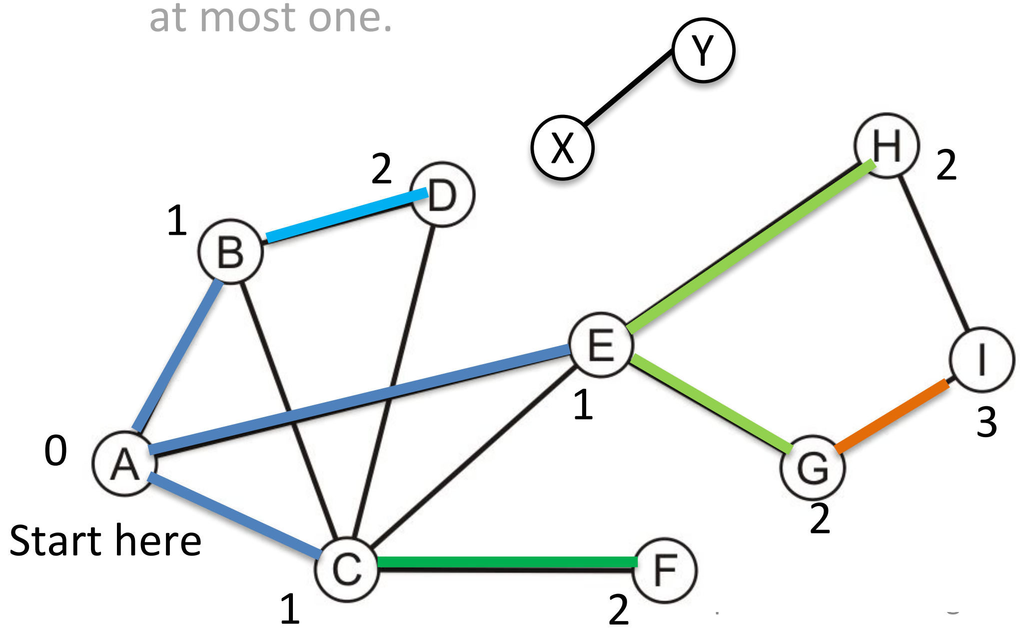
Properties of BFS

- Let G be an undirected graph on which BFS traversal starting at vertex s has been performed.
- Then
  - BFS traversal visits all vertices in the connected component that has vertex s.
  - Vertices are visited in order of “level” or how far they are from s
  - The discovery-edges form a spanning tree T of the connected component that contains s. We call this “BFS tree” of the connected component
  - For each vertex v at level i, the path of the BFS tree T between s and v has i edges, and any other path of G between s and v has at least i edges.
  - If (u, v) is an edge that is not in the BFS tree, then the level numbers of u and v differ by at most one.



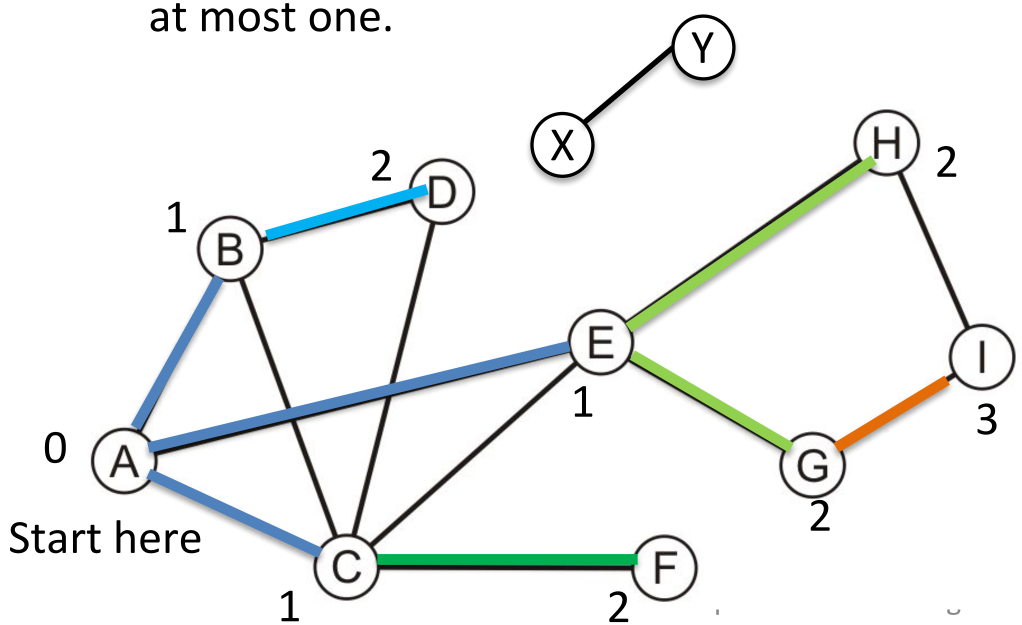
Properties of BFS

- Let G be an undirected graph on which BFS traversal starting at vertex s has been performed.
- Then
  - BFS traversal visits all vertices in the connected component that has vertex s.
  - Vertices are visited in order of "level" or how far they are from s
  - The discovery-edges form a spanning tree T. We call this "BFS tree" of the connected component that has vertex s
  - For each vertex v at level i, the path of the BFS tree T between s and v has i edges, and any other path of G between s and v has at least i edges.
  - If (u, v) is an edge that is not in the BFS tree, then the level numbers of u and v differ by at most one.



Properties of BFS

- Let G be an undirected graph on which BFS traversal starting at vertex s has been performed.
- Then
  - BFS traversal visits all vertices in the connected component that has vertex s.
  - Vertices are visited in order of “level” or how far they are from s
  - The discovery-edges form a spanning tree T. We call this “BFS tree” of the connected component that has vertex s
  - For each vertex v at level i, the path of the BFS tree T between s and v has i edges, and any other path of G between s and v has at least i edges.
  - If (u, v) is an edge that is not in the BFS tree, then the level numbers of u and v differ by at most one.



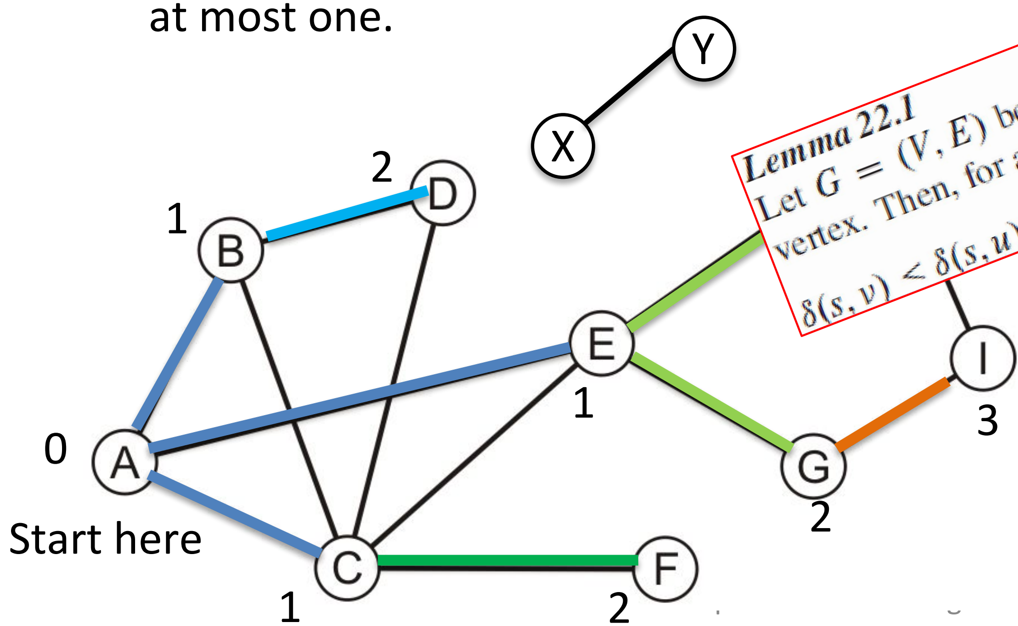
Properties of BFS

- Let G be an undirected graph on which BFS traversal starting at vertex s has been performed.
- Then
  - BFS traversal visits all vertices in the connected component that has vertex s.
  - Vertices are visited in order of “level” or how far they are from s
  - The discovery-edges form a spanning tree T. We call this “BFS tree” of the connected component that has vertex s
  - For each vertex v at level i, the path of the BFS tree T between s and v ha… any other path of G between s and v has at least i edges.
  - If (u, v) is an edge that is not in the BFS tree, then the lev… …er by at most one.

Lemma 22.1
Let $G = (V, E)$ be a directed or undirected graph, and let $s \in V$ be an arbitrary vertex. Then, for any edge $(u, v) \in E$,

$$\delta(s, v) \le \delta(s, u) + 1 .$$

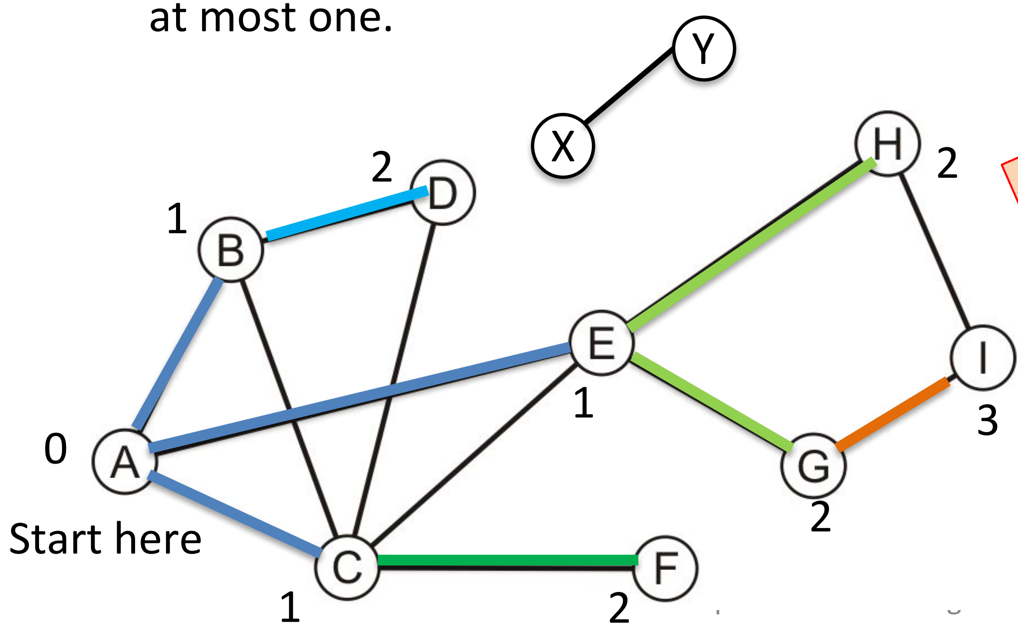
δ(u,v) is the path between u and v with fewest no of edges



Properties of BFS

- Let G be an undirected graph on which BFS traversal starting at vertex s has been performed.
- Then
  - BFS traversal visits all vertices in the connected component that has vertex s.
  - Vertices are visited in order of "level" or how far they are from s
  - The discovery-edges form a spanning tree T. We call this "BFS tree" of the connected component that has vertex s
  - For each vertex v at level i, the path of the BFS tree T between s and v has i edges, and any other path of G between s and v has at least i edges.
  - If (u, v) is an edge that is not in the BFS tree, then the level numbers ffer by at most one.

Indeed v.d = δ(s, v). That is BFS computes the shortest path (with fewest edges) between S and v



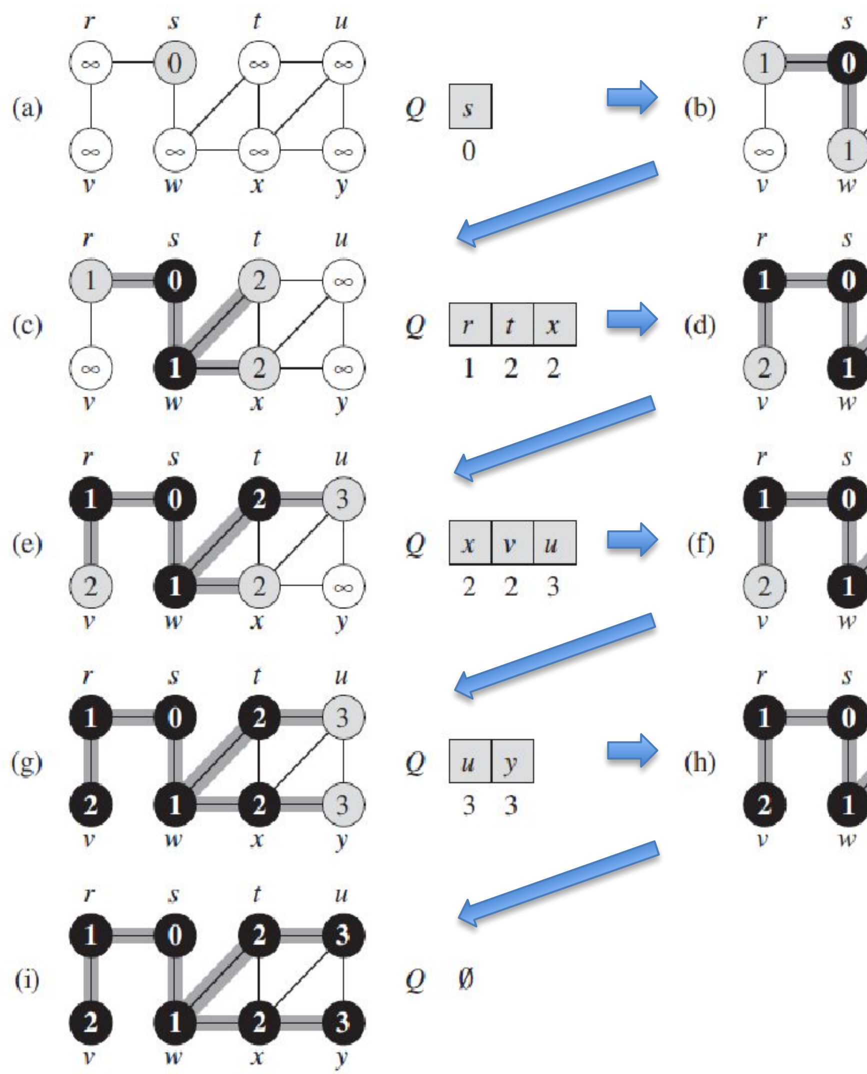
Time complexity of BFS

- Let $G = (V, E)$ be an undirected graph , and $|V| = n$, $|E| = m$.
- Then BFS of G starting at vertex s takes time $O(n + m)$
- How about $\Omega(f(n, m))$ or $\theta(g(n, m))$?

- Also, there exist $O(n + m)$ time algorithms based on BFS for the following problems:
  - Testing whether G is connected
  - Computing a spanning tree of G
  - Computing the connected components of G
  - Computing, for every vertex v of G, the minimum number of edges of any path between s and v.

Breadth-First Search

- Time complexity of BFS: O(|V| + |E|)



Max no. of OPS

|V|-1 = O(|V|)

|V|-1 = O(|V|)

1 = O(1)

|V|+1 = O(|V|)

|V| = O(|V|)

|V|+|E| = O(|V| + |E|)

|E| = O(|E|)

|E| = O(|E|)

- Identify the starting vertex, s
- Initialize:

for each vertex $u \in G.V - \{s\}$
    $u.color = \text{WHITE}$
    $u.d = \infty$
    $u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$
$Q = \emptyset$
$\text{ENQUEUE}(Q, s)$

- Then:

while $Q \neq \emptyset$
    $u = \text{DEQUEUE}(Q)$
    for each $v \in G.Adj[u]$
        if $v.color == \text{WHITE}$
            $v.color = \text{GRAY}$
            $v.d = u.d + 1$
            $v.\pi = u$
            $\text{ENQUEUE}(Q, v)$
    $u.color = \text{BLACK}$

Time complexity of BFS

- Let $G = (V, E)$ be an undirected graph , and $|V| = n$, $|E| = m$.
- Then BFS of G starting at vertex s takes time $O(n + m)$
- How about $\Omega(f(n, m))$ or $\theta(g(n, m))$?

- Also, there exist $O(n + m)$ time algorithms based on BFS for the following problems:
  - Testing whether G is connected
  - Computing a spanning tree of G
  - Computing the connected components of G
  - Computing, for every vertex v of G, the minimum number of edges of any path between s and v.

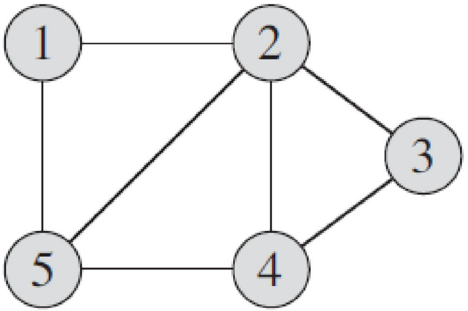
Time complexity of BFS

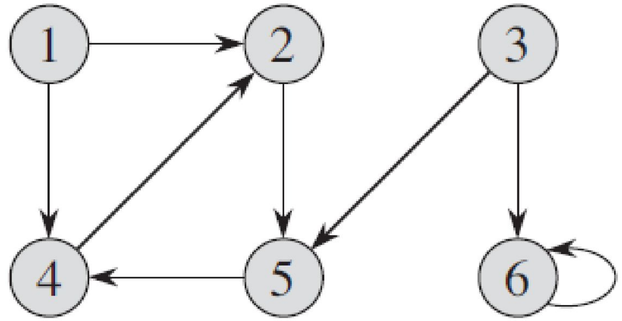
- Let $G = (V, E)$ be an undirected graph , and $|V| = n$, $|E| = m$.
- Then BFS of G starting at vertex s takes time $O(n + m)$
- How about $\Omega(f(n, m))$ or $\theta(g(n, m))$?

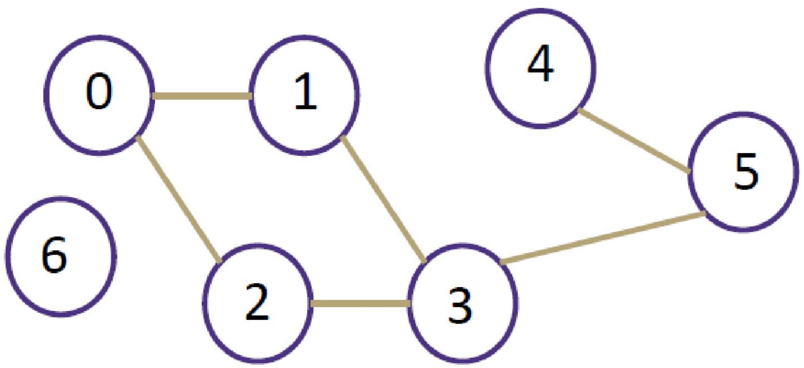
- Also, there exist $O(n + m)$ time algorithms based on BFS for the following problems:
  – Testing whether G is connected
  – Computing a spanning tree of G
  – Computing the connected components of G
  – Computing, for every vertex v of G, the minimum number of edges of any path between s and v.

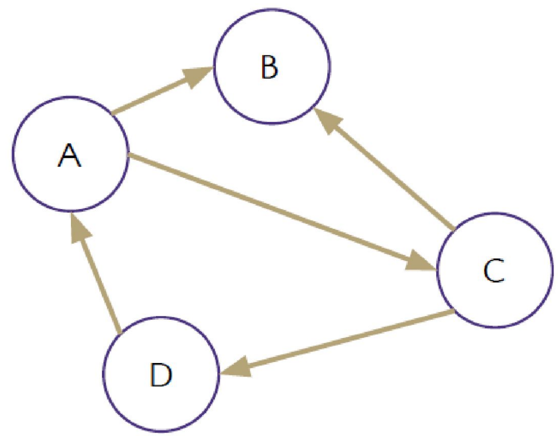
Try these examples

BFS traversal of the following graphs starting with vertex 2, for example



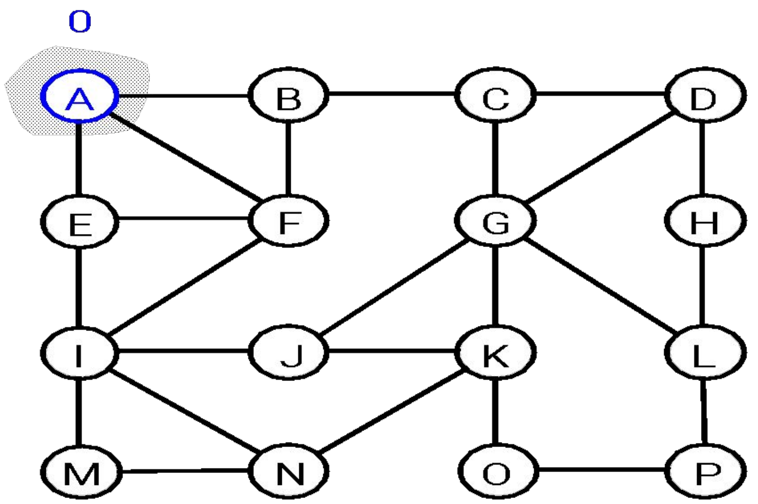


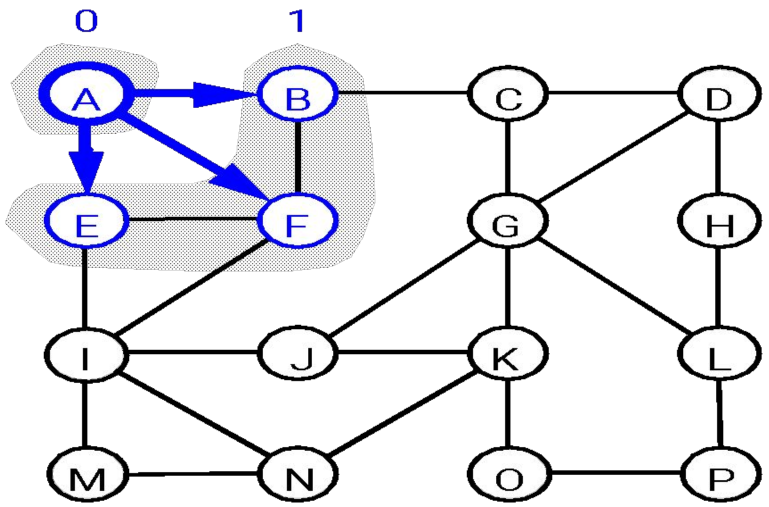




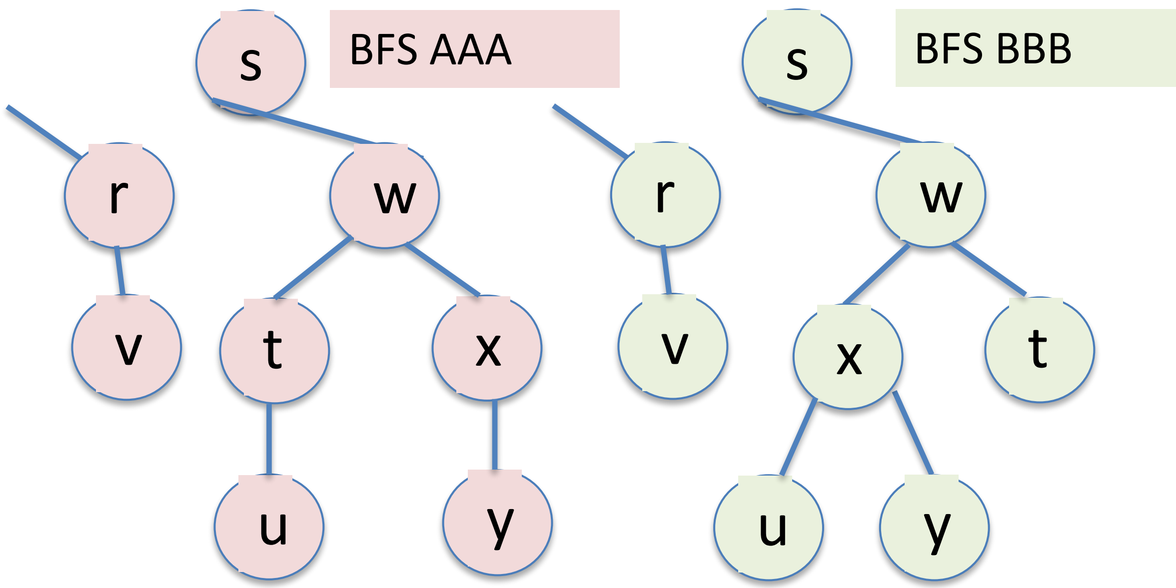
Try these examples

Here is another example to try out:





Q&A


s
BFS AAA
r
w
v
t
x
u
y
s
BFS BBB
r
w
v
x
t
u
y