# Advanced Algorithms Analysis and Design

By

Nazir Ahmad Zafar

# Lecture No 28

## Breadth First Search

# Contents of the Today Lecture

- Representation of Graphs
- Breadth First Search
  - Algorithm
  - Analysis
  - Supporting lemmas in the proof
  - Proof of correctness
  - Shortest paths, for un-weighted edges, based on Breadth First Search
- Conclusion

# Representations of Graphs

- Two standard ways to represent a graph
  - Adjacency lists,
  - Adjacency Matrix
- Applicable to directed and undirected graphs.

Adjacency lists

- A compact way to represent **sparse graphs**.
  - $|E|$ is much less than $|V|^2$
- Graph $G(V, E)$ is represented by array *Adj* of $|V|$ lists
- For each $u \in V$, the adjacency list $Adj[u]$ consists of all the vertices adjacent to $u$ in $G$
- The amount of memory required is: $(V + E)$

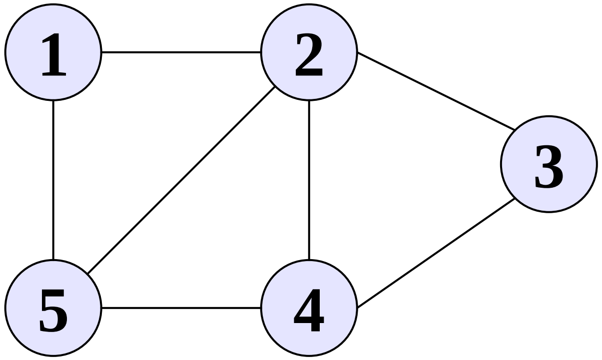
# Adjacency Matrix

- A graph $G(V, E)$ assuming the vertices are numbered 1, 2, 3, … , $|V|$ in some arbitrary manner, then representation of $G$ consists of: $|V| \times |V|$ matrix A = $(a_{ij})$ such that

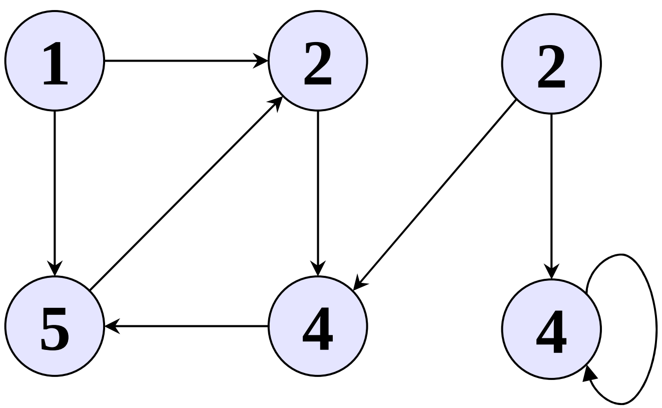
$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in \mathrm{E} \\ 0 & \text{otherwise} \end{cases}$$

- Preferred when graph is **dense**
  - $|E|$ is close to $|V|^2$

# Adjacency matrix of undirected graph




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




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

# Adjacency Matrix

- The amount of memory required is

$$\Theta(V^2)$$

- For undirected graph to cut down needed memory only entries on and above diagonal are saved
  - In an undirected graph, *(u, v)* and *(v, u)* represents the same edge, adjacency matrix A of an undirected graph is its own **transpose**

$$\mathbf{A = A^T}$$

- It can be adapted to represent **weighted graphs**.

# Breadth First Search

# Breadth First Search

- One of **simplest** algorithm searching graphs
- A vertex is **discovered** first time, encountered
- Let *G (V, E)* be a graph with **source** vertex *s*, BFS
  - discovers every vertex reachable from *s.*
  - gives distance from *s* to each reachable vertex
  - produces BF tree root with *s* to reachable vertices
- To keep track of progress, it colors each vertex
  - vertices start **white,** may later **gray,** then **black**
  - Adjacent to black vertices have been discovered
  - Gray vertices may have some adjacent white vertices

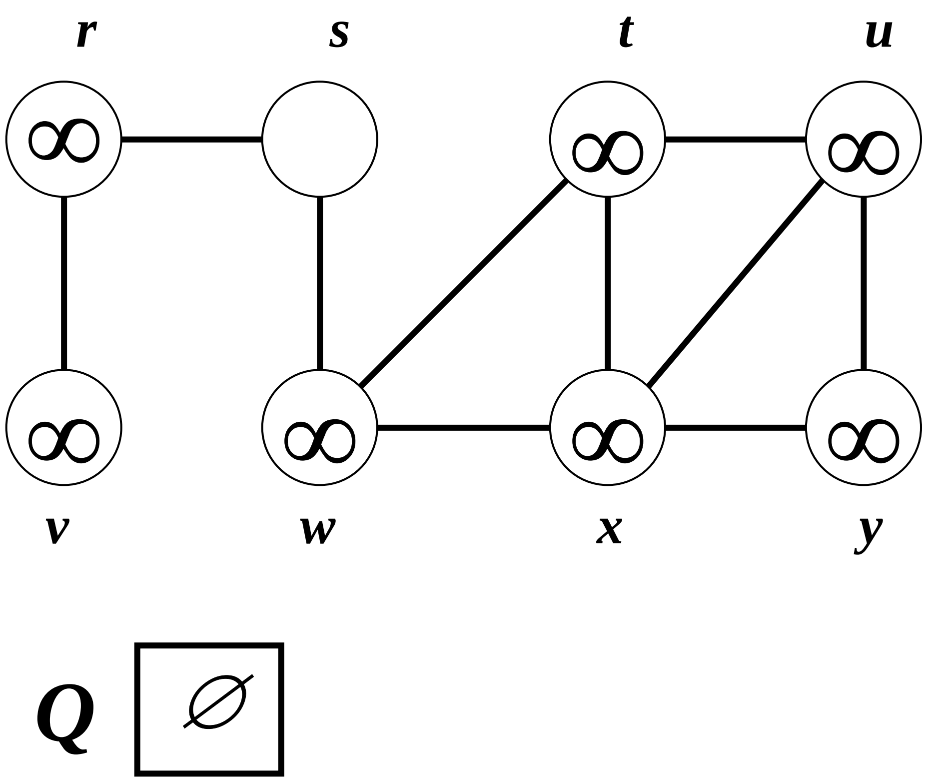
# Breadth First Search

- It is assumed that input graph *G (V, E)* is represented using adjacency list.

- Additional structures maintained with each vertex $v \in V$ are
  - *color[u]* – stores color of each vertex
  - $\pi[u]$ – stores predecessor of *u*
  - *d[u]* – stores distance from source *s* to vertex *u*

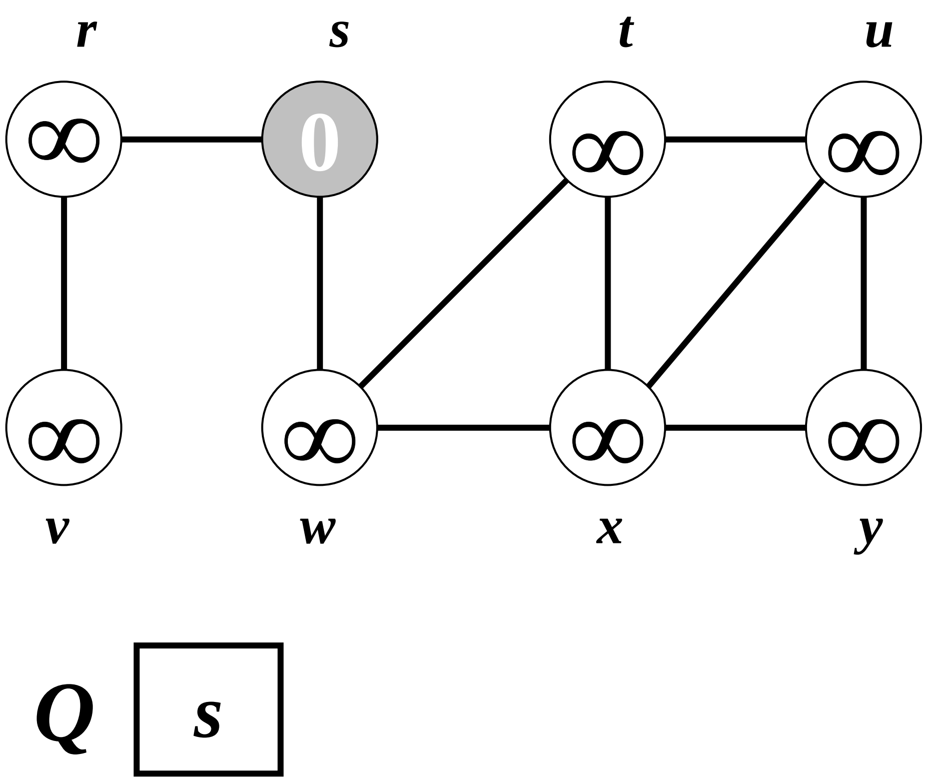
# Breadth First Search

```
BFS(G, s)
  for each vertex u ∈ V [G] – {s}
      do color [u] ← WHITE
         d [u] ← ∞
         π[u] ← NIL
  color[s] ← GRAY
  d [s] ← 0
  π[s] ← NIL
  Q ← Ø                      /* Q always contains the set of GRAY vertices */
  ENQUEUE (Q, s)
  while Q ≠ Ø
      do u ← DEQUEUE (Q)
         for each v ∈ Adj [u]
             do if color [v] = WHITE          /* For undiscovered vertex. */
                   then color [v] ← GRAY
                        d [v] ← d [u] + 1
                        π[v] ← u
                        ENQUEUE(Q, v)
         color [u] ← BLACK
```




Except root node, *s*
For each vertex u $\in$ *V(G)*
*color* [*u*] $\leftarrow$ WHITE
*d*[*u*] $\leftarrow \infty$
$\pi$ [*s*] $\leftarrow$ *NIL*

# Breadth First Search



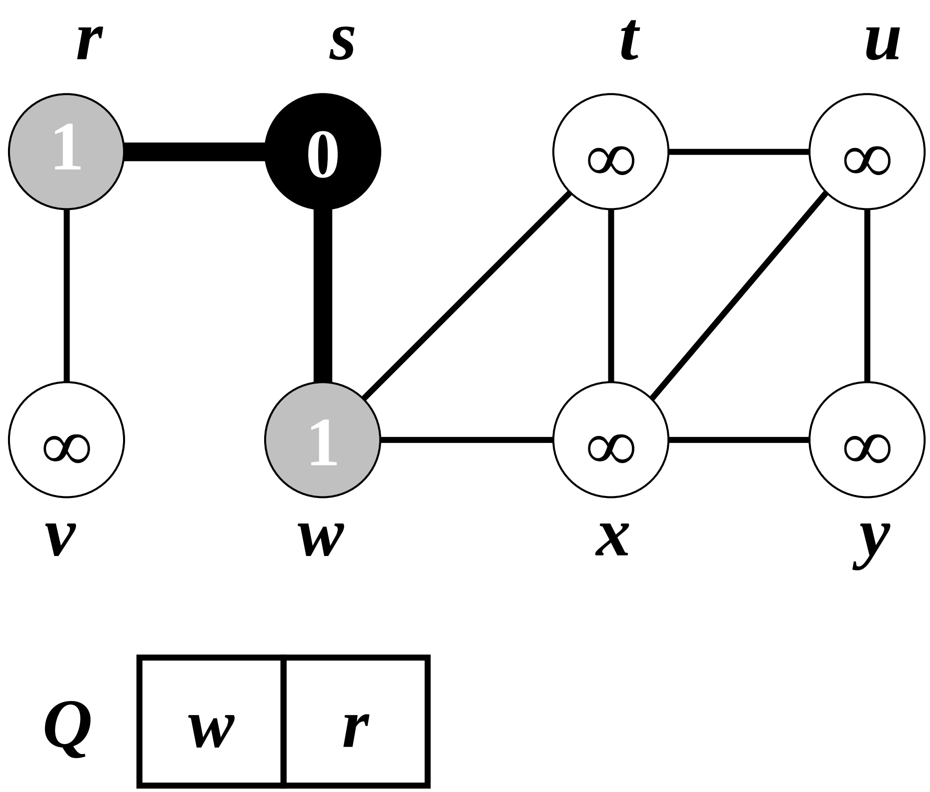

Considering *s* as root node

$d[s] \leftarrow 0$

$\pi\ [s] \leftarrow NIL$

ENQUEUE (*Q*, *s*)

# Breadth First Search




DEQUEUE $s$ from Q

$Adj[s] = w, r$

$color\ [w]$ = WHITE

$color\ [w] \leftarrow$ GRAY

$d\ [w] \leftarrow d\ [s] + 1 = 0 + 1 = 1$

$\pi[w] \leftarrow s$

ENQUEUE $(Q,\ w)$

$color\ [r]$ = WHITE

$color\ [r] \leftarrow$ GRAY

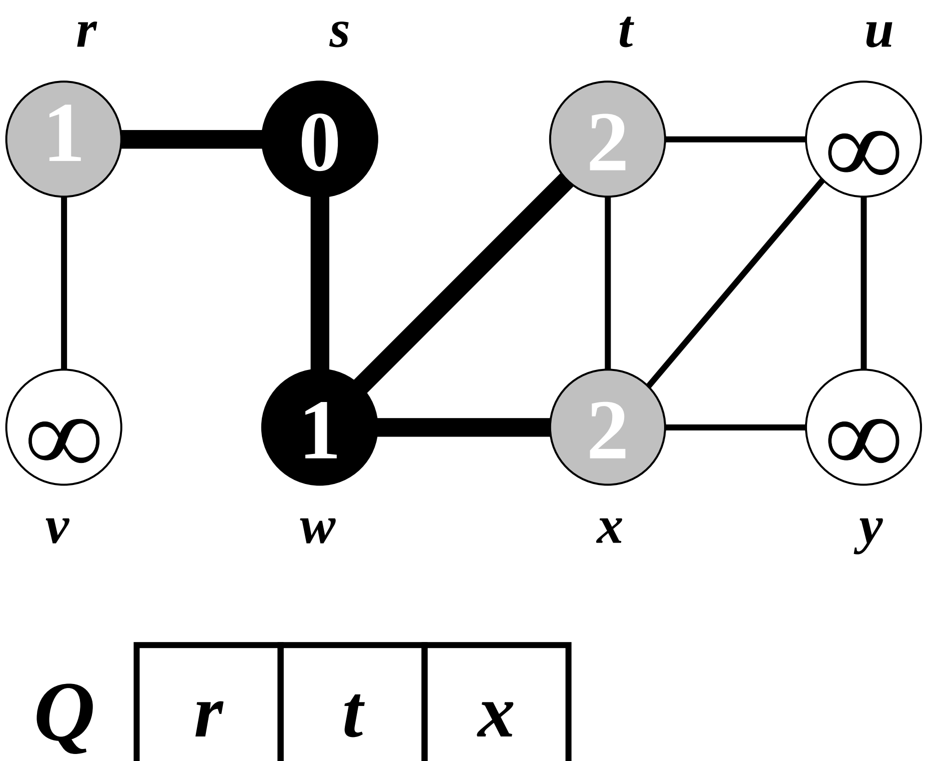
$d\ [r] \leftarrow d\ [s] + 1 = 0 + 1 = 1$

$\pi[r] \leftarrow s$

ENQUEUE $(Q,\ r)$

$color\ [s] \leftarrow$ BLACK

# Breadth First Search




DEQUEUE $w$ from Q

$Adj[w] = s, t, x$

$color\ [s] \neq$ WHITE

$color\ [t] =$ WHITE

$color\ [t] \leftarrow$ GRAY

$d\ [t] \leftarrow d\ [w] + 1 = 1 + 1 = 2$

$\pi[t] \leftarrow w$

ENQUEUE ($Q$, $t$)

$color\ [x] =$ WHITE

$color\ [x] \leftarrow$ GRAY

$d\ [x] \leftarrow d\ [w] + 1 = 1 + 1 = 2$

$\pi[x] \leftarrow w$

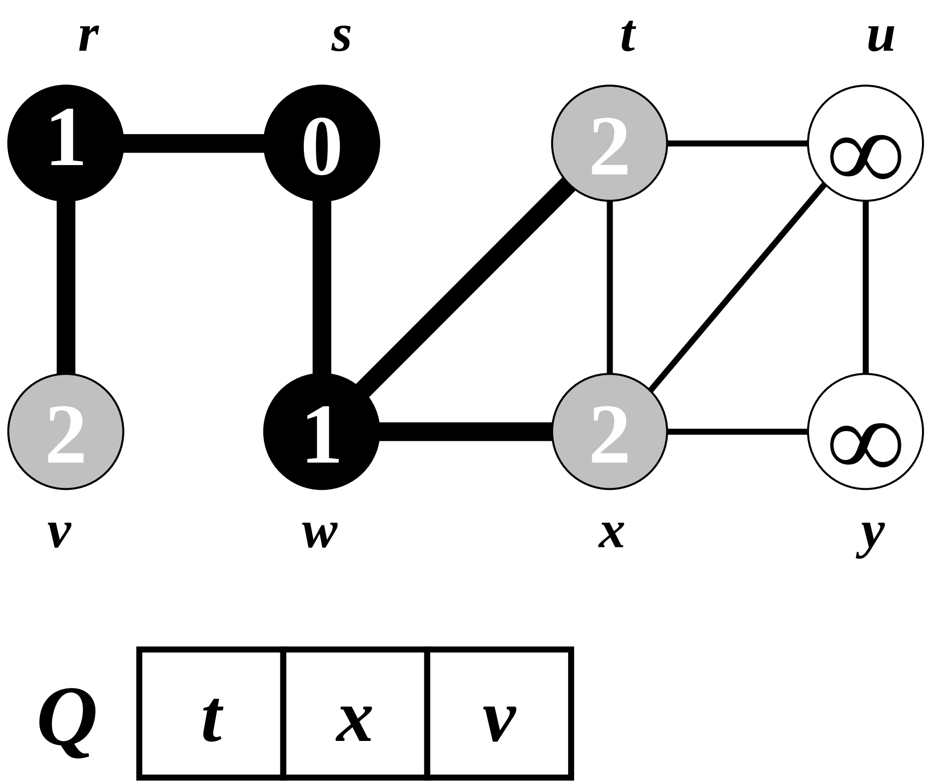
ENQUEUE ($Q$, $x$)

$color\ [w] \leftarrow$ BLACK

# Breadth First Search

DEQUEUE $r$ from Q

$Adj[r] = s, v$




$color\ [s] \neq$ WHITE

$color\ [v] =$ WHITE

$color\ [v] \leftarrow$ GRAY

$d\ [v] \leftarrow d\ [r] + 1 = 1 + 1 = 2$

$\pi[v] \leftarrow r$

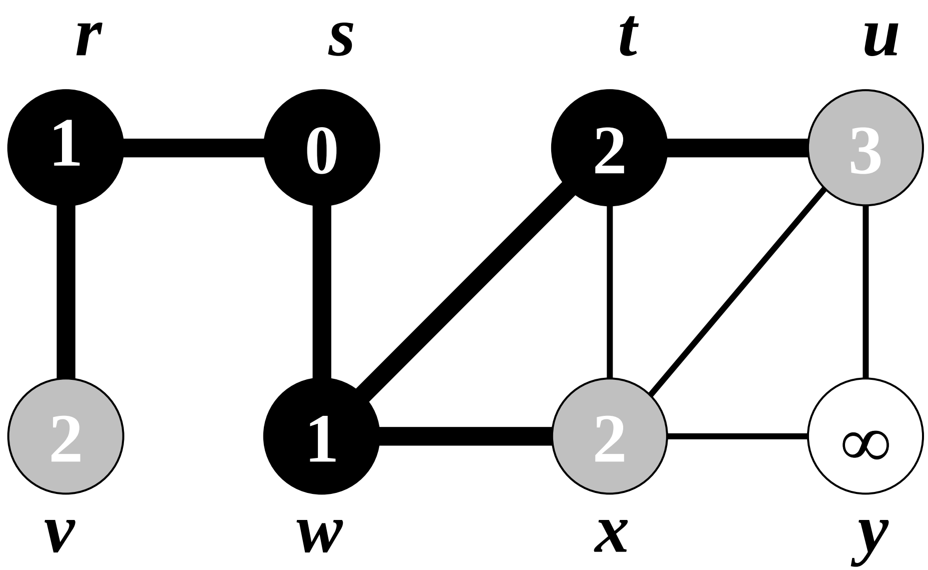
ENQUEUE ($Q$, $v$)

$color\ [r] \leftarrow$ BLACK

# Breadth First Search

DEQUEUE $t$ from Q

$Adj[t] = u, w, x$




| Q | | | |
|---|---|---|---|
| Q | $x$ | $v$ | $u$ |

$color\ [u] = \text{WHITE}$

$color\ [u] \leftarrow \text{GRAY}$

$d\ [u] \leftarrow d\ [t] + 1 = 2 + 1 = 3$

$\pi[u] \leftarrow t$

ENQUEUE $(Q,\ u)$

$color\ [w] \neq \text{WHITE}$

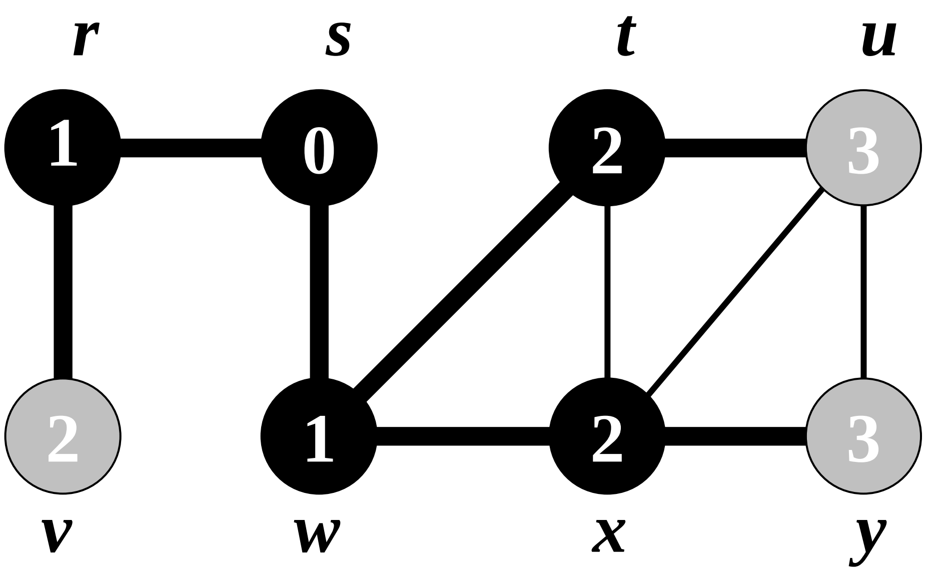
$color\ [x] \neq \text{WHITE}$

$color\ [t] \leftarrow \text{BLACK}$

# Breadth First Search

DEQUEUE $x$ from Q

$Adj[x] = t, u, w, y$




$color\ [t] \neq$ WHITE

$color\ [u] \neq$ WHITE

$color\ [w] \neq$ WHITE

$color\ [y] =$ WHITE

$color\ [y] \leftarrow$ GRAY

$d\ [y] \leftarrow d\ [x] + 1 = 2 + 1 = 3$

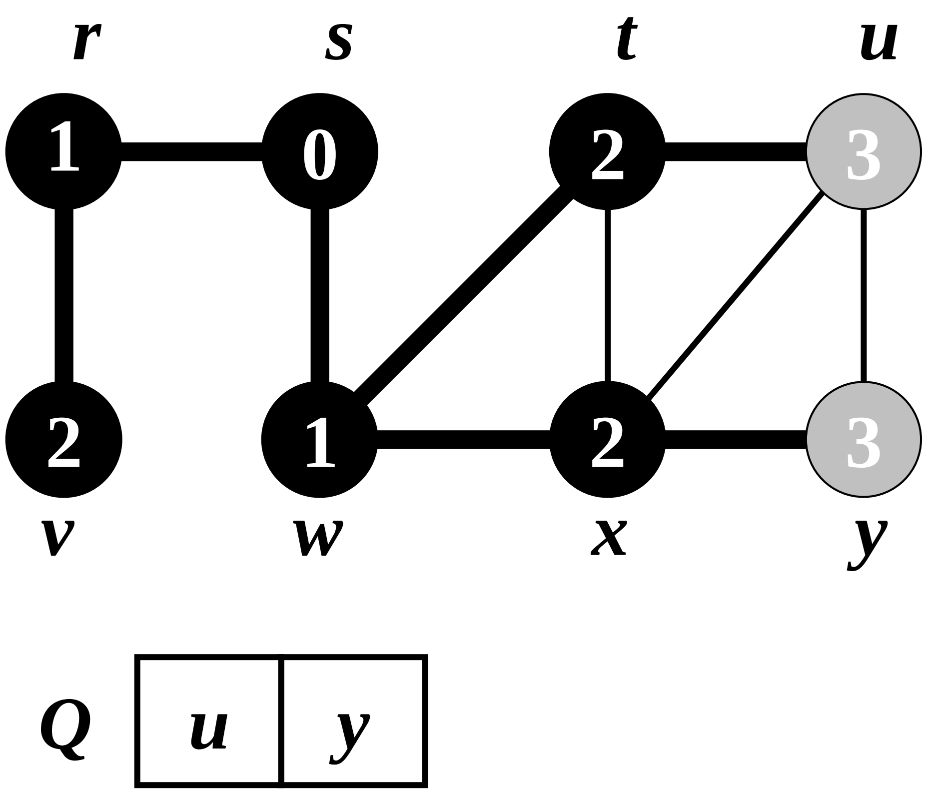
$\pi[y] \leftarrow x$

ENQUEUE $(Q,\ y)$

| **Q** | | | |
|---|---|---|---|
| | *v* | *u* | *y* |

$color\ [x] \leftarrow$ BLACK

# Breadth First Search

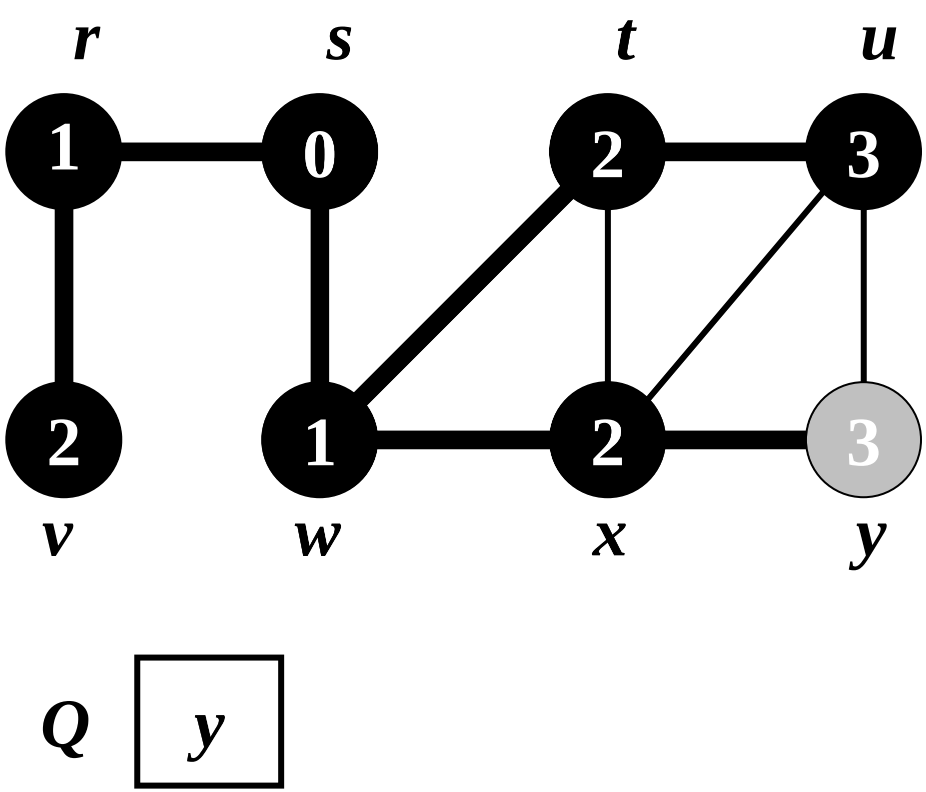



DEQUEUE $v$ from Q
$Adj[v] = r$

$color\ [r] \neq$ WHITE

$color\ [v] \leftarrow$ BLACK

# Breadth First Search




DEQUEUE *u* from Q

*Adj[u]* = *t*, *x*, *y*

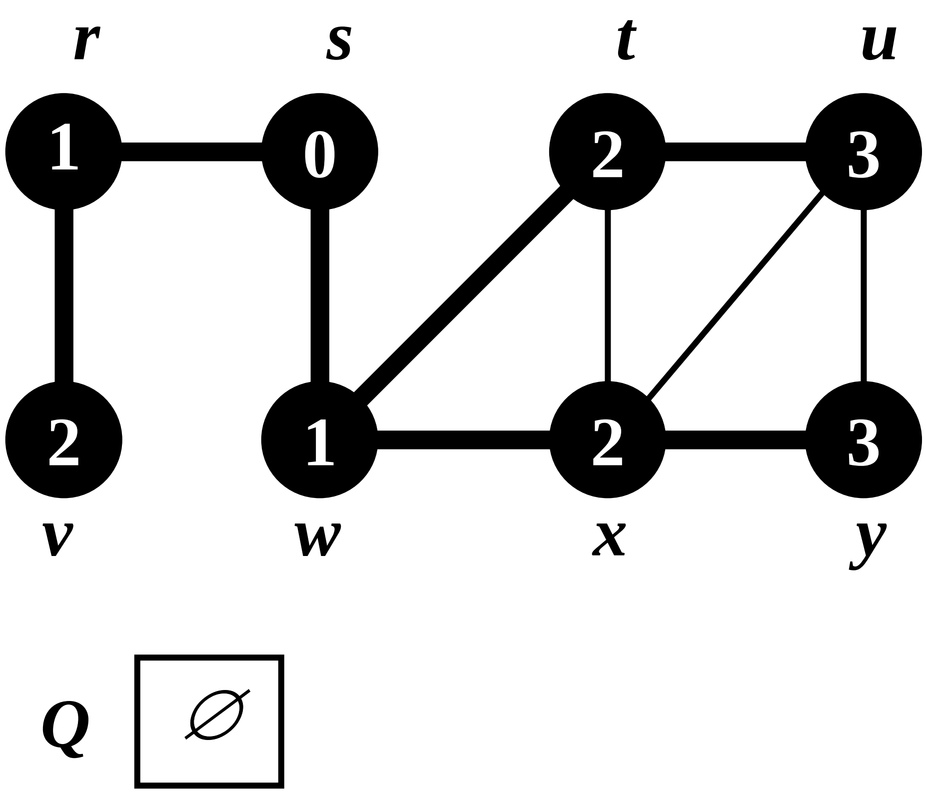
*color* [*t*] ≠ WHITE

*color* [*x*] ≠ WHITE

*color* [*y*] ≠ WHITE

*color* [*u*] ← BLACK

# Breadth First Search




DEQUEUE $y$ from Q

$Adj[y] = u, x$

$color\ [u] \neq$ WHITE

$color\ [x] \neq$ WHITE

$color\ [y] \leftarrow$ BLACK

# Breadth First Search

- Each vertex is enqueued and dequeued atmost once
  - Total time devoted to queue operation is $O(V)$
- The sum of lengths of all adjacency lists is $\Theta(E)$
  - Total time spent in scanning adjacency lists is $O(E)$
- The overhead for initialization $O(V)$

**Total Running Time of BFS = $O(V+E)$**

# Shortest Paths

- The **shortest-path-distance** $\delta\ (s,\ v)$ from s to v as the minimum number of edges in any path from vertex $s$ to vertex $v$.
  - if there is no path from $s$ to $v$, then $\delta\ (s,\ v) = \infty$
- A path of length $\delta\ (s,\ v)$ from $s$ to $v$ is said to be a **shortest path** from $s$ to $v$.
- Breadth First search finds the distance to each reachable vertex in the graph $G\ (V,\ E)$ from a given source vertex $s \in V$.
- The field $d$, for distance, of each vertex is used.

# Shortest Paths

**BFS-Shortest-Paths** (G, s)

```
∀ v ∈ V
     d [v] ← ∞
d [s] ← 0
ENQUEUE (Q, s)
while Q ≠ φ
      do v ← DEQUEUE(Q)
           for each w in Adj[v]
                do if d [w] = ∞
                      then d [w] ← d [v] +1
                           ENQUEUE (Q, w)
```

# Lemma

## Statement:

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

$$\delta(s, v) \leq \delta(s, u) + 1$$

## Proof

- If $u$ is reachable from $s$, then so is $v$. In this case, the shortest path from $s$ to $v$ cannot be longer than the shortest path from $s$ to $u$ followed by the edge $(u, v)$, and thus the inequality holds.
- If $u$ is not reachable from $s$, then $\delta(s, u) = \infty$, and the inequality holds.

# Lemma

Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on $G$ from a given source vertex $s \in V$. Then upon termination, for each vertex $v \in V$, value $d[v]$ computed by BFS satisfies:

$$d[v] \geq \delta(s, v).$$

Proof

- We use induction on the number of ENQUEUE operations.
  - Inductive Hypothesis $d[v] \geq \delta(s, v)$ for all $v \in V$
- The basis of the induction is the situation immediately after $s$ is enqueued in line 9 of BFS.

# Lemma (contd..)

- The inductive hypothesis holds here, because $d[s] = 0 = \delta(s, s)$ and $d[v] = \infty \geq \delta(s, v)$ for all $v \in V - \{s\}$
- For inductive step, consider a white vertex $v$ that is discovered during the search from a vertex $u$.
- By Inductive hypothesis: $d[u] \geq \delta(s, u)$. By line 15 and from previous Lemma, we obtain

$$d[v] = d[u] + 1 \geq \delta(s, u) + 1 \geq \delta(s, v)$$

- Vertex $v$ is then enqueued, and it is never enqueued again because it is also grayed and the then clause of lines 14-17 is executed only for white vertices.
- Thus, the value of $d[v]$ never changes again, and the inductive hypothesis is maintained

# Lemma

## Statement

- Suppose that during execution of BFS on a graph $G = (V, E)$, the queue $Q$ contains the vertices $\langle v_1, v_2, \ldots, v_r \rangle$, *where* $v_1$ is the head of $Q$ and $v_r$ is the tail. Then, $d[v_r] \leq d[v_1] + 1$ and $d[v_i] \leq d[v_{i+1}]$ for $i = 1, 2, \ldots, r - 1$

## Proof

- – The proof is by induction on the number of queue operations
- – Initially, when the queue contains only $s$, the lemma certainly holds

# Lemma (contd..)

- For the inductive step, we must prove that the lemma holds after both dequeuing and enqueuing a vertex.
- If the head $v_1$ of the queue is dequeued, $v_2$ becomes the new head. (If the queue becomes empty, then the lemma holds vacuously.)

$$d[v_1] \leq d[v_2] \quad \text{(The inductive hypothesis)}$$

- But then we have $d[v_r] \leq d[v_1] + 1 \leq d[v_2] + 1$ and the remaining inequalities are unaffected.
- Enqueuing vertex requires close examination of code.
- When we enqueue a vertex $v$ in line 17 of BFS, it becomes $v_{r+1}$.

# Lemma (contd..)

- At that time, we have already removed vertex $u$, whose adjacency list is currently being scanned, from the queue $Q$, and by the inductive hypothesis, the new head $v_1$ has $d[v_1] \geq d[u]$.
- Thus, $d[v_{r+1}] = d[v] = d[u] + 1 \leq d[v_1] + 1$ (The inductive hypothesis),
- we also have $d[v_r] \leq d[u] + 1$, and so $d[v_r] \leq d[u] + 1 = d[v] = d[v_{r+1}]$, and the remaining inequalities are unaffected.
- Thus, the lemma follows when $v$ is enqueued

# Corollary

## Statement:

Suppose that vertices $v_i$ and $v_j$ are enqueued during execution of BFS, and that $v_i$ is enqueued before $v_j$. Then $d[v_i] \leq d[v_j]$ at the time that $v_j$ *is* enqueued.

## Proof

- Immediate from above Lemma and
- the property that each vertex receives a finite *d* value at most once during the course of BFS

# Theorem (Correctness of BFS)

## Statement

Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on $G$ from a given source vertex $s \in V$. Then, during its execution, BFS discovers every vertex $v \in V$ that is reachable from the source $s$, and upon termination

$$d[v] = \delta(s, v) \text{ for all } v \in V.$$

Moreover, for any vertex $v \neq s$ that is reachable from $s$, one of the shortest paths from $s$ to $v$ is a shortest path from $s$ to $\pi[v]$ followed by edge $(\pi[v], v)$.

# Theorem (Correctness of BFS)

## Proof

- Assume, for the purpose of contradiction, that some vertex receives a *d* value not equal to its shortest path distance.

- Let $v$ be the vertex with minimum $\delta(s, v)$ that receives such an incorrect *d* value; clearly $v \neq s$.

- By Lemma 22.2, $d[v] \geq \delta(s, v)$, and thus we have that $d[v] > \delta(s, v)$. Vertex $v$ must be reachable from $s$, for if it is not, then $\delta(s, v) = \infty \geq d[v]$.

# Theorem (Correctness of BFS)

- Let $u$ be the vertex immediately preceding $v$ on a shortest path from $s$ to $v$, so that

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

- Because $\delta(s, u) < \delta(s, v)$, and because of how we chose $v$, we have $d[u] = \delta(s, u)$.

- Putting these properties together, we have

$$d[u] > \delta(s, v) = \delta(s, u) + 1 = d[u] + 1 \qquad (22.1)$$

- Now consider the time when BFS chooses to dequeue vertex $u$ from $Q$ in line 11.

# Theorem (Correctness of BFS)

- At this time, vertex *v* is, white, gray, or black.
- We shall show that in each of these cases, we derive a contradiction to inequality (22.1).
- If *v* is white, then line 15 sets $d[v] = d[u] + 1$, contradicting inequality (22.1).
- If *v* is black, then it was already removed from the queue and, by Corollary 22.4, we have $d[v] \leq d[u]$, again contradicting inequality (22.1).

- If *v* is gray, then it was painted gray upon dequeuing some vertex *w*, which was removed from *Q* earlier than *u* and, $d[v] = d[w] + 1$.

# Theorem (Correctness of BFS)

- By Corollary 22.4, however, $d[w] \leq d[u]$, and so we have $d[v] \leq d[u] + 1$, once again contradicting inequality (22.1).
- Thus we conclude that $d[v] = \delta(s, v)$ for all $v \in V$. All vertices reachable from $s$ must be discovered, if they were not, they would have infinite $d$ values.
- To conclude the proof of the theorem, observe that if $\pi[v] = u$, then $d[v] = d[u] + 1$.
- Thus, we can obtain a shortest path from $s$ to $v$ by taking a shortest path from $s$ to $\pi[v]$ and then traversing the edge $(\pi[v], v)$

# Lemma

## Statement

When applied to a directed or undirected graph $G = (V, E)$, procedure BFS constructs $\pi$ so that the predecessor subgraph $G_\pi = (V_\pi, E_\pi)$ is a breadth-first tree.

## Proof

- Line 16 of BFS sets $\pi[v] = u$ if and only if $(u\ v) \in E$ and $\delta(s, v) < \infty$ that is, if $v$ is reachable from $s$ and thus $V_\pi$ consists of the vertices in $V$ reachable from $s$.

# Lemma

- Since $G_\pi$ forms a tree, it contains a unique path from $s$ to each vertex in $V_\pi$ .

- By applying previous Theorem inductively, we conclude that every such path is a shortest path.

- The procedure in upcoming slide prints out the vertices on a shortest path from $s$ to $v$, assuming that BFS has already been run to compute the shortest-path tree.

# Print Path

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

# Conclusion

- How graphs can be represented
- Breadth First Search Techniques is discussed
- Algorithms is designed
- It is to be noted that just designing an algorithm of any problem is not enough, to give its proof is required as well.
- Correctness of Breadth First Search is given
- BFS algorithm is refined to find shortest path
- This shortest path is, of course, for un-weighted graphs
- Searching algorithms have various applications.