

*Elementary*  
*Graph Algorithms*

**PART-2**

1

**Outlines: Graphs Part-2**

---

- Graph Search Methods
- Breadth-First Search (BFS):
  - **BFS Algorithm**
  - **BFS Example**
  - **BFS Time Complexity**
  - **Output of BFS:**
    - **Shortest Path**
    - **Breath-First Tree**
    - **Is the Graph Connected?**

2

## Graph Search Methods

---

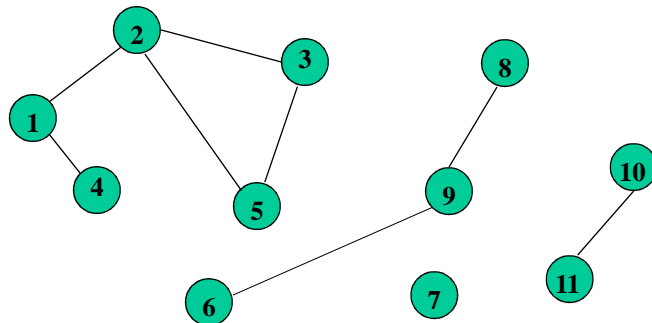
- Many graph problems solved using a search method
  - Path from one vertex to another
  - Is the graph connected?
  - etc.
- Commonly used search methods:
  - Breadth-First Search (BFS)
  - Depth-First Search (DFS)

3

## Graph Search Methods

---

- A vertex  $u$  is reachable from vertex  $v$  iff there is a path from  $v$  to  $u$ .
- A search method starts at a given vertex  $v$  and visits every vertex that is reachable from  $v$ .



4

## Graph Search Methods

---

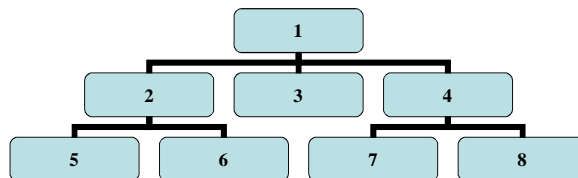
- Given: a graph  $G = (V, E)$ , directed or undirected
- Goal: methodically **explore every vertex** and **every edge**
- Ultimately: build a **tree** on the graph
  - Pick a vertex as the root
  - Choose certain edges to produce a *tree*
  - Note: might also build a *forest* if graph is not connected

5

## Breadth First Search

---

- Example: Binary Tree (This is a special case of a graph).
  - The order of search is across levels.
  - The root is examined first; then children of the root; then the children of those nodes, and so on.



6

## Breadth First Search

---

- Example: Directed Graph
- Pick a source vertex  $\mathcal{S}$  to start.
- Find (or discover) the vertices that are adjacent to  $\mathcal{S}$ .
- Pick each child of  $\mathcal{S}$  in turn and discover their vertices adjacent to that child.
- Done when all children have been discovered and examined.
- This results in a tree that is rooted at the source vertex  $\mathcal{S}$ .
- The idea is to find the *distance* from the selected Source vertex to all reachable vertices (Destinations).

7

## Breadth-First Search

---

- Visit start vertex (s) and put into a **FIFO** queue.
- Repeatedly remove a vertex from the queue, visit its unvisited adjacent vertices, put newly visited vertices into the queue.
- All vertices reachable from the start vertex (s) (including the start vertex) are visited.

8

## Breadth-First Search

---

- Again will associate vertex “colors” to guide the algorithm
  - **White** vertices have not been discovered
    - All vertices start out white
  - **Gray** vertices are discovered but not fully explored
    - They may be adjacent to white vertices
  - **Black** vertices are discovered and fully explored
    - They are adjacent only to black and gray vertices
- Explore vertices by scanning adjacency list of gray vertices

9

## Breadth-First Search

---

```
BFS(G, s) {
  // initialize vertices;
  1 for each u ∈ V(G) - {s} {
  2   do color[u] = WHITE
  3     d[u] = ∞           // distance from s to u
  4     p[u] = NIL       // predecessor or parent of u
  }
  5 color[s] = GRAY
  6 d[s] = 0
  7 p[s] = NIL
  8 Q = Empty;
  9 Enqueue(Q, s);      // Q is a queue; initialize to s
 10 while (Q not empty) {
 11   u = Dequeue(Q);
 12   for each v ∈ adj[u] {
 13     if (color[v] == WHITE)
 14       color[v] = GRAY;
 15       d[v] = d[u] + 1;
 16       p[v] = u;      What does d[v] represent?
 17       Enqueue(Q, v); What does p[v] represent?
 18   } color[u] = BLACK;
  }
}
```

10

## Breadth-First Search

---

- Lines 1-4 paint every vertex white, set  $d[u]$  to be infinity for each vertex  $(u)$ , and set  $p[u]$  the parent of every vertex to be NIL.
- Line 5 paints the source vertex  $(s)$  gray.
- Line 6 initializes  $d[s]$  to 0.
- Line 7 sets the parent of the source to be NIL.
- Lines 8-9 initialize  $Q$  to the queue containing just the vertex  $(s)$ .
- The while loop of lines 10-18 iterates as long as there remain gray vertices, which are discovered vertices that have not yet had their adjacency lists fully examined.
  - This while loop maintains the test in line 10, the queue  $Q$  consists of the set of the gray vertices.

11

## Breadth-First Search

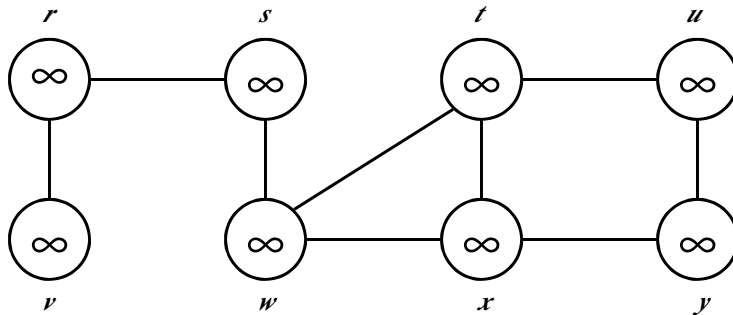
---

- Prior to the first iteration in line 10, the only gray vertex, and the only vertex in  $Q$ , is the source vertex  $(s)$ .
- Line 11 determines the gray vertex  $(u)$  at the head of the queue  $Q$  and removes it from  $Q$ .
- The for loop of lines 12-17 considers each vertex  $(v)$  in the adjacency list of  $(u)$ .
- If  $(v)$  is white, then it has not yet been discovered, and the algorithm discovers it by executing lines 14-17.
  - It is first grayed, and its distance  $d[v]$  is set to  $d[u]+1$ .
  - Then,  $u$  is recorded as its parent.
  - Finally, it is placed at the tail of the queue  $Q$ .
- When all the vertices on  $(u)$ 's adjacency list have been examined,  $u$  is blackened in line 18.

12

## Breadth-First Search: Example

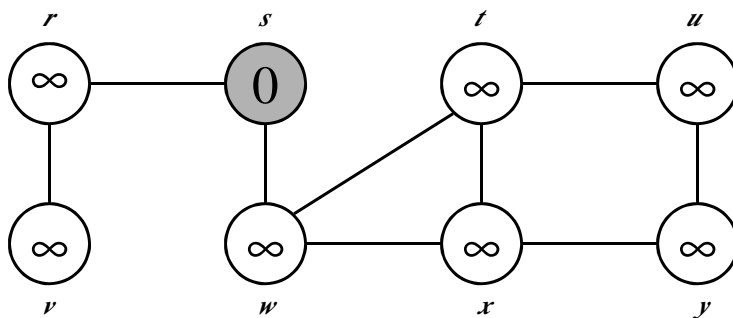
---



13

## Breadth-First Search: Example

---

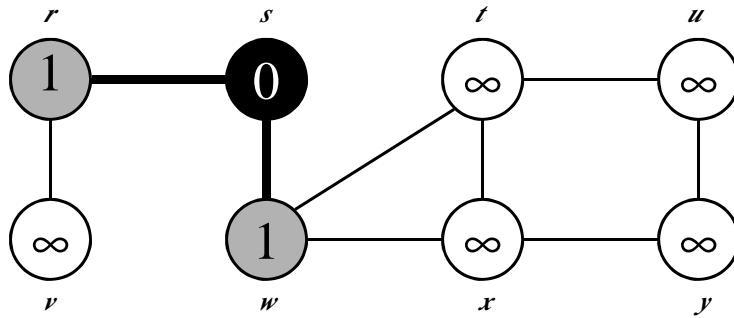


$Q: \boxed{s}$

14

## Breadth-First Search: Example

---

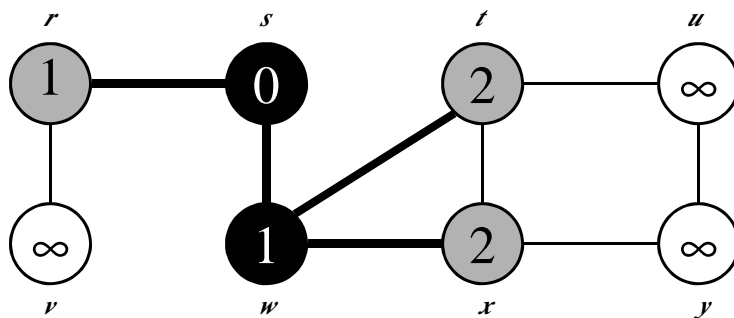


$Q: \begin{array}{|c|c|} \hline w & r \\ \hline \end{array}$

15

## Breadth-First Search: Example

---



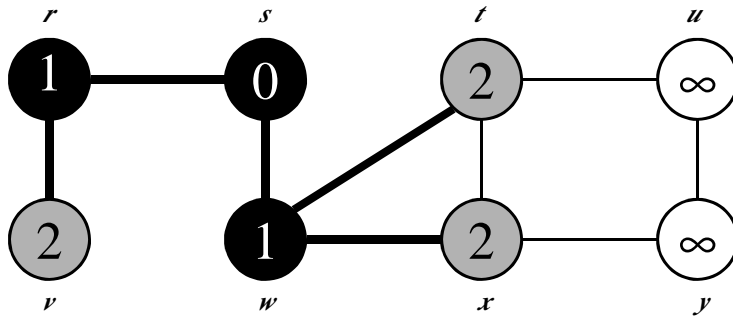
$Q: \begin{array}{|c|c|c|} \hline r & t & x \\ \hline \end{array}$

16



## Breadth-First Search: Example

---

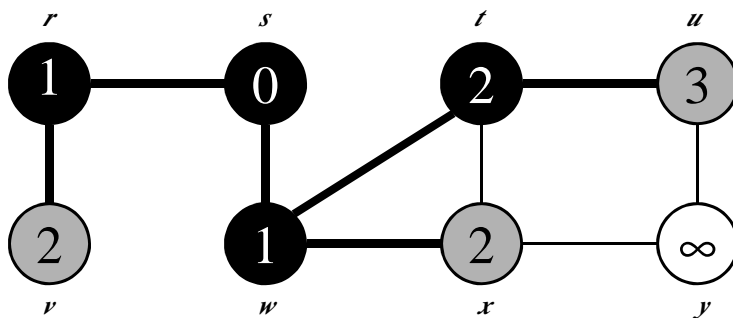


$Q: \begin{array}{|c|c|c|} \hline t & x & v \\ \hline \end{array}$

17

## Breadth-First Search: Example

---

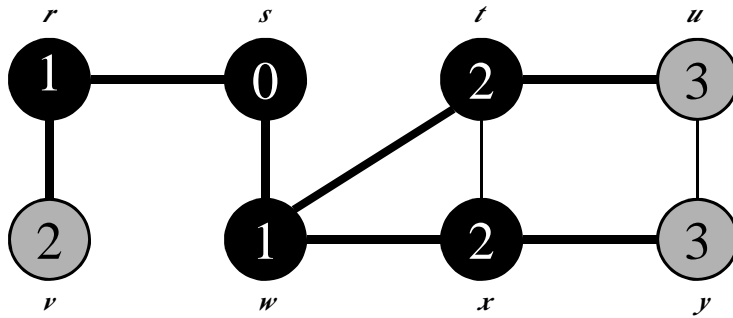


$Q: \begin{array}{|c|c|c|} \hline x & v & u \\ \hline \end{array}$

18

## Breadth-First Search: Example

---



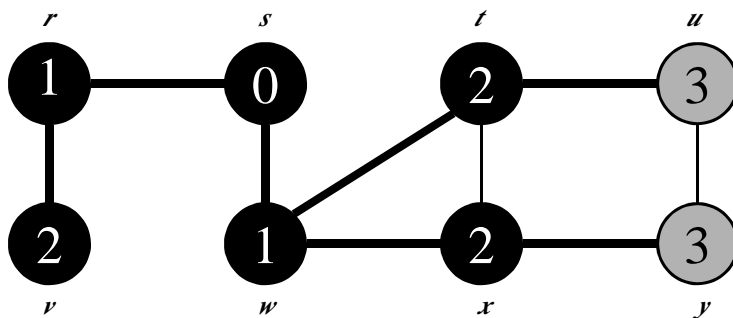
$Q$ : 

$v$	$u$	$y$
-----	-----	-----

19

## Breadth-First Search: Example

---



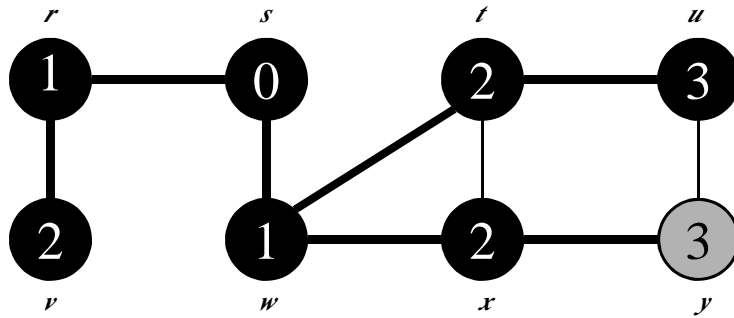
$Q$ : 

$u$	$y$
-----	-----

20

## Breadth-First Search: Example

---

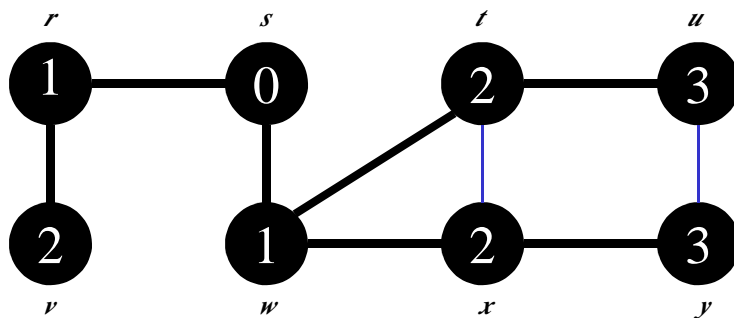


$Q: \boxed{y}$

21

## Breadth-First Search: Example

---



$Q: \emptyset$

22

## BFS: The Code Again

```

BFS(G, s) {
  // initialize vertices
  for each u ∈ V(G) - {s} { ← Touch every vertex: O(V)
    do color[u] = WHITE
       d[u] = ∞
       p[u] = NIL
  }
  color[s] = GRAY;
  d[s] = 0;
  p[s] = NIL;
  Q = Empty;
  Enqueue(Q, s);
  while (Q not empty) {
    u = Dequeue(Q); ← u = every vertex, but only once
    for each v ∈ adj[u] {
      if (color[v] == WHITE)
        color[v] = GRAY;
        d[v] = d[u] + 1;
        p[v] = u;
        Enqueue(Q, v);
    }
    color[u] = BLACK;
  }
}

```

So  $v =$  every vertex that appears in some other vertex's adjacency list ←  $O(E)$

What will be the running time?  
Total running time:  $O(V+E)$

23

## BFS: Time Complexity

- Given a graph  $G = (V, E)$ 
  - Vertices are enqueued if their color is white
  - Assuming that en- and dequeuing takes  $O(1)$  time the total cost of this operation is  $O(V)$
  - Adjacency list of a vertex is scanned when the vertex is dequeued (and at most once)
  - The sum of the lengths of all lists is  $O(E)$ . Consequently,  $O(E)$  time is spent on scanning them
  - Initializing the algorithm takes  $O(V)$
- **Total running time  $O(V+E)$** 
  - linear in the size of the adjacency list representation of  $G$

24

## BFS: The Code Again

```

BFS(G, s) {
  // initialize vertices
  for each u ∈ V(G) - {s} { Each vertex will be enqueue once.
    do color[u] = WHITE
       d[u] = ∞
       p[u] = NIL
  }
  color[s] = GRAY
  d[s] = 0
  p[s] = NIL
  Q = Empty;
  Enqueue (Q,s);
  while (Q not empty) {
    u = Dequeue(Q);
    for each v ∈ adj[u] {
      if (color[v] == WHITE) What will be the storage cost
        color[v] = GREY; in addition to storing the tree?
        d[v] = d[u] + 1; Queue Size
        p[v] = u; Total space used:
        Enqueue(Q, v); at least = O(max(degree(v)))
    } at most = O(V)
    color[u] = BLACK;
  }
}

```

25

## Shortest Paths

- Given a graph  $G = (V, E)$ , BFS discovers all vertices reachable from a source vertex  $s$ 
  - It computes the *shortest-path distance* to all reachable vertices
  - It computes a *breadth-first tree* that contains all such reachable vertices
- For any vertex  $v$  reachable from  $s$ , the path in the breadth first tree from  $s$  to  $v$ , corresponds to a **shortest path** in  $G$ .
- BFS calculates the *shortest-path distance* to the source node
  - shortest-path distance = minimum number of edges from  $s$  to  $v = \delta(s, v) = d[v]$
  - or  $\delta(s, v) = \infty$  if  $v$  not reachable from  $s$

26

## Breadth-First Tree

---

- BFS builds *breadth-first tree*, in which paths to root represent shortest paths in  $G$
- Given a graph  $G = (V, E)$  with source  $s \in V$ , we define the predecessor subgraph of  $G$  as  $G_p = (V_p, E_p)$ , where
  - $V_p = \{v \in V : p[v] \neq \text{NIL}\} \cup \{s\}$
  - $E_p = \{(p[v], v) : v \in V_p - \{s\}\}$
- $G_p$  is a breadth-first tree
  - $V_p$  consists of the vertices reachable from  $s$ , and
  - for all  $v \in V_p$ , there is a unique simple path from  $s$  to  $v$  in  $G_p$  that is also a shortest path from  $s$  to  $v$  in  $G$ .
- The edges in  $G_p$  are called tree edges

27

## Path From Vertex $v$ To Vertex $u$

---

- Start a breadth-first search at vertex  $v$ .
- Terminate when vertex  $u$  is visited or when  $Q$  becomes empty (whichever occurs first).
- Time  $O(V+E)$

28

## Is The Graph Connected?

---

- Start a breadth-first search at any vertex of the graph.
- Graph is connected iff all  $n$  vertices get visited.
- Time  $O(V+E)$