CS 4407
Algorithms

Lecture 6:
Basic Graph Search Algorithms

Prof. Gregory Provan
Department of Computer Science
University College Cork

Outline

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

Today's Learning Objectives

- Introduce the two basic graph search algorithms
- Show the applications of these algorithms

Important Task: Finding Shortest Paths

- Given an undirected graph and source vertex s, the length of a path in a graph (without edge weights) is the number of edges on the path.
- Problem: Find the shortest path from s to each other vertex in the graph.
- Brute-force method: enumerate all simple paths starting from s and keep track of the shortest path arriving at each vertex. There may be $n!$ simple paths in a graph….

Breadth-First-Search (BFS)

- Given:
  - $G = (V, E)$.
  - A distinguished *source* vertex.
- Systematically explore the edges of G to discover every vertex that is reachable from s:
  - Compute the (shortest) distance from s to all reachable vertices.
  - Produce a *breadth-first tree* with root s that contains all reachable vertices.

Breadth-First-Search (BFS)

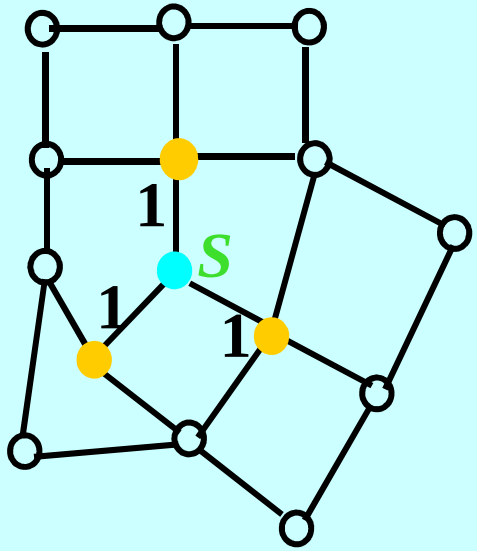
- **BFS colors each vertex:**

  white -- undiscovered

  gray -- discovered but "not done yet"

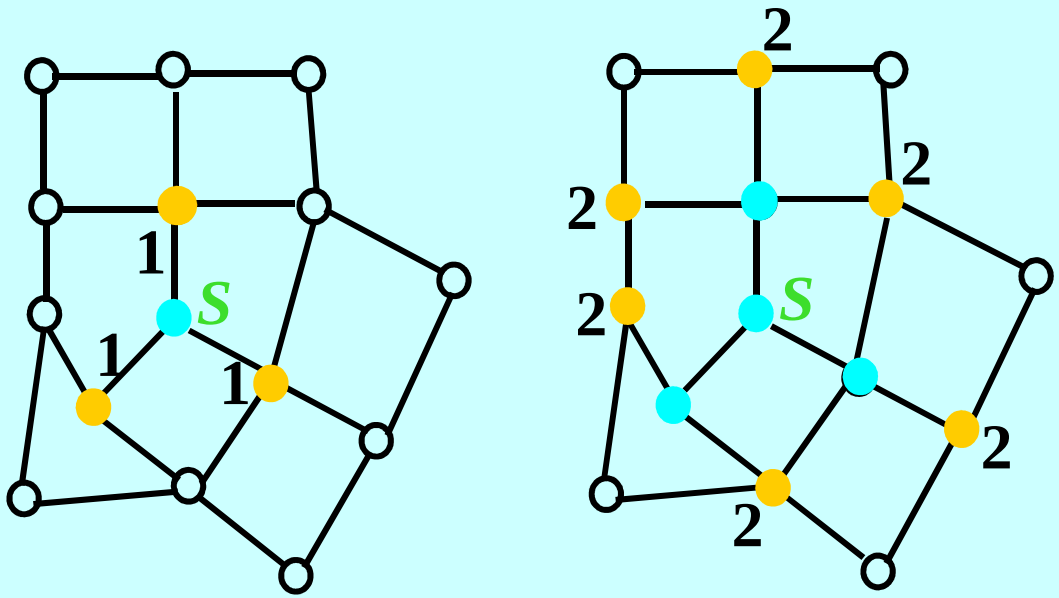
  black -- all adjacent vertices have been discovered

BFS for Shortest Paths



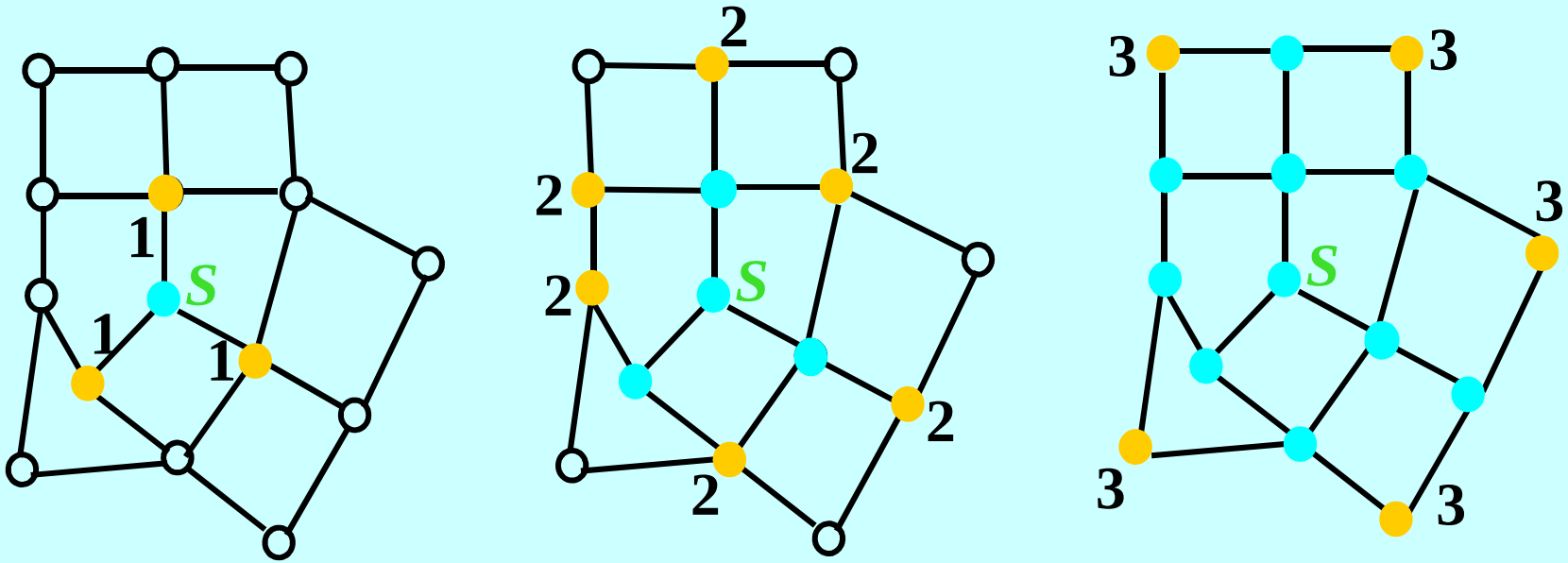
● Finished ● Discovered ○ Undiscovered

BFS for Shortest Paths



● Finished ● Discovered ○ Undiscovered

BFS for Shortest Paths



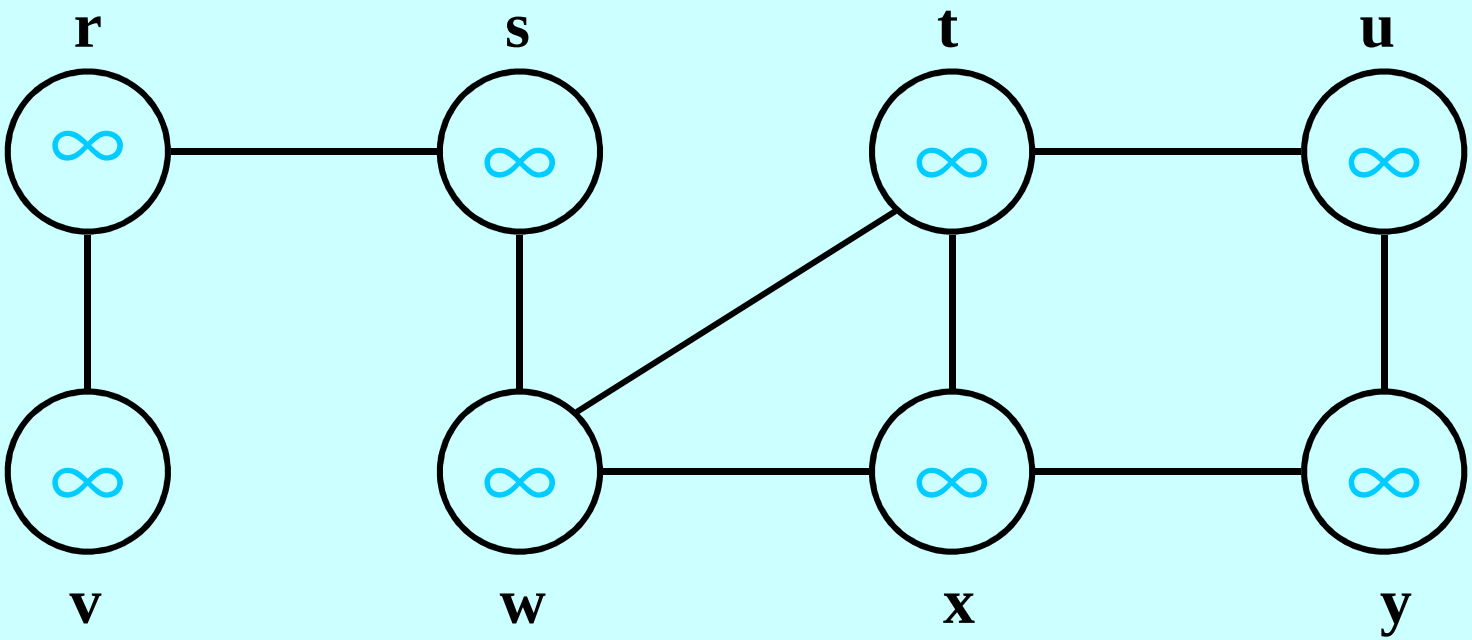
● Finished ● Discovered ○ Undiscovered

BFS(G, s)

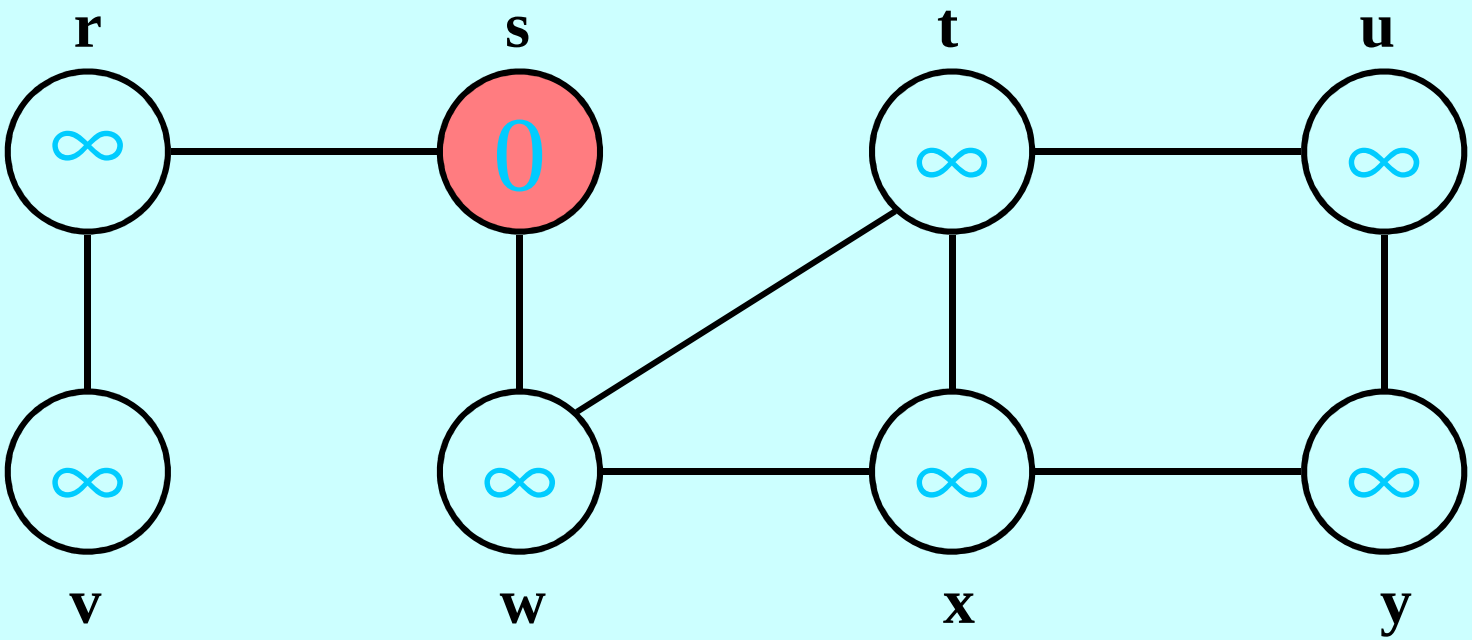
```
1.   for each vertex u ∈ (V[G] – {s})
2.         do color[u] ← White
3.         d[u] ← ∞
4.         π[u] ← Nil
5.   color[s] ← Gray
6.   d[s] ← 0
7.   π[s] ← Nil
8.   Q ← ∅
9.   Enqueue(Q, s)
10.  while Q ≠ ∅
11.        do u ← Dequeue(Q)
12.              for each v ∈ Adj[u]
13.                    do if color[v] = White
14.                          then color[v] ← Gray
15.                          d[v] ← d[u] + 1
16.                          π[v] ← u
17.                          Enqueue(Q, v)
18.              color[u] ← Black
```

Q: a queue of discovered vertices
$color[v]$: color of v
$d[v]$: distance from s to v
$\pi[v]$: predecessor of v

Breadth-First Search: Example

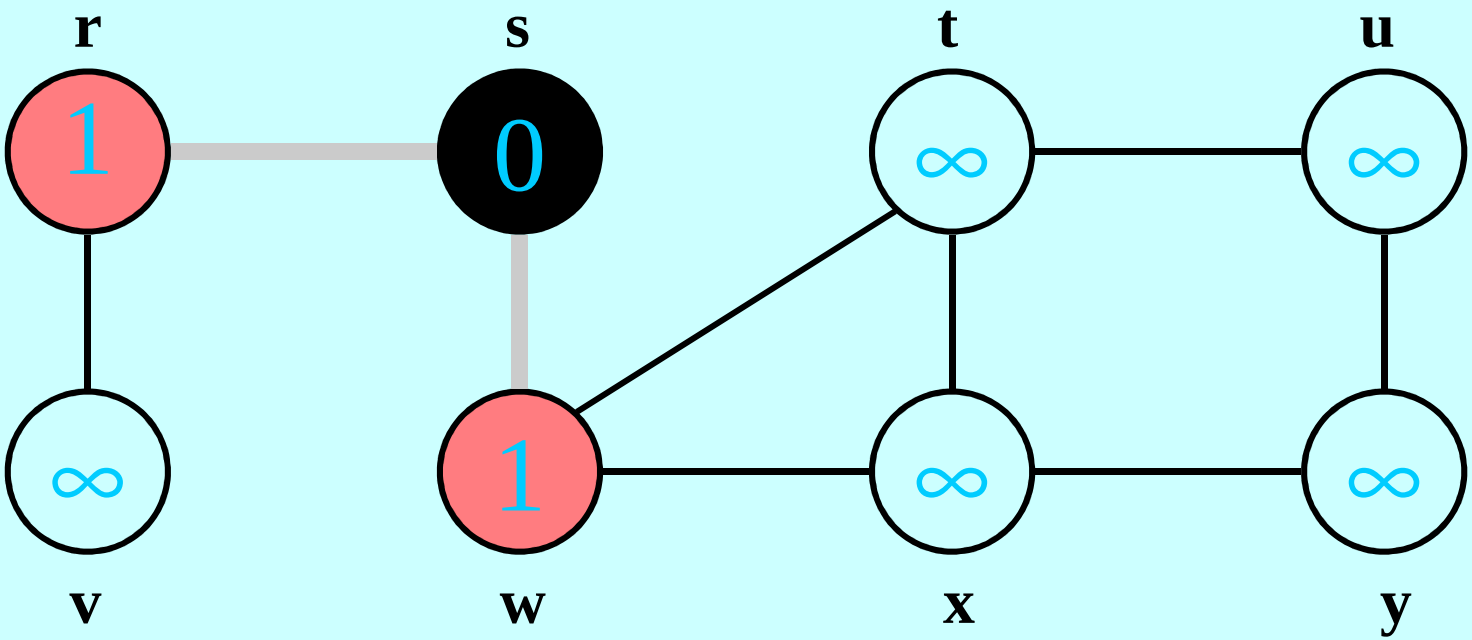


Breadth-First Search: Example



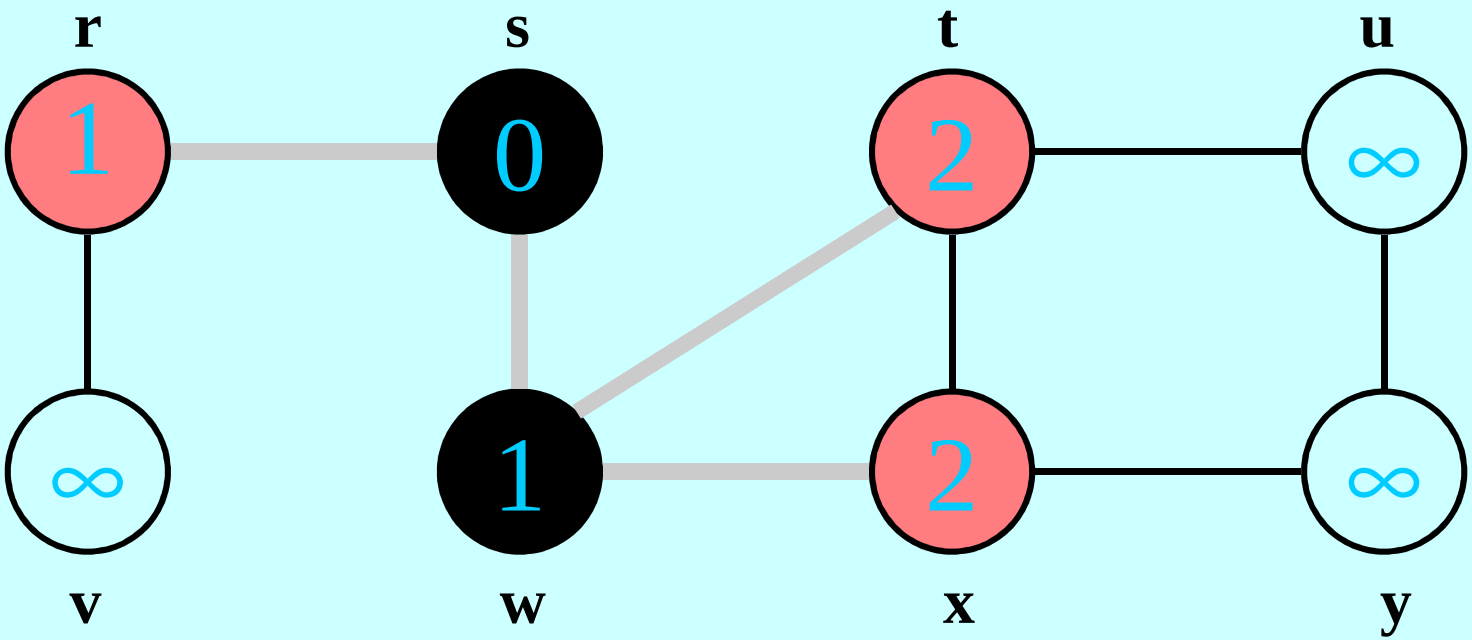
Q: | s |

Breadth-First Search: Example



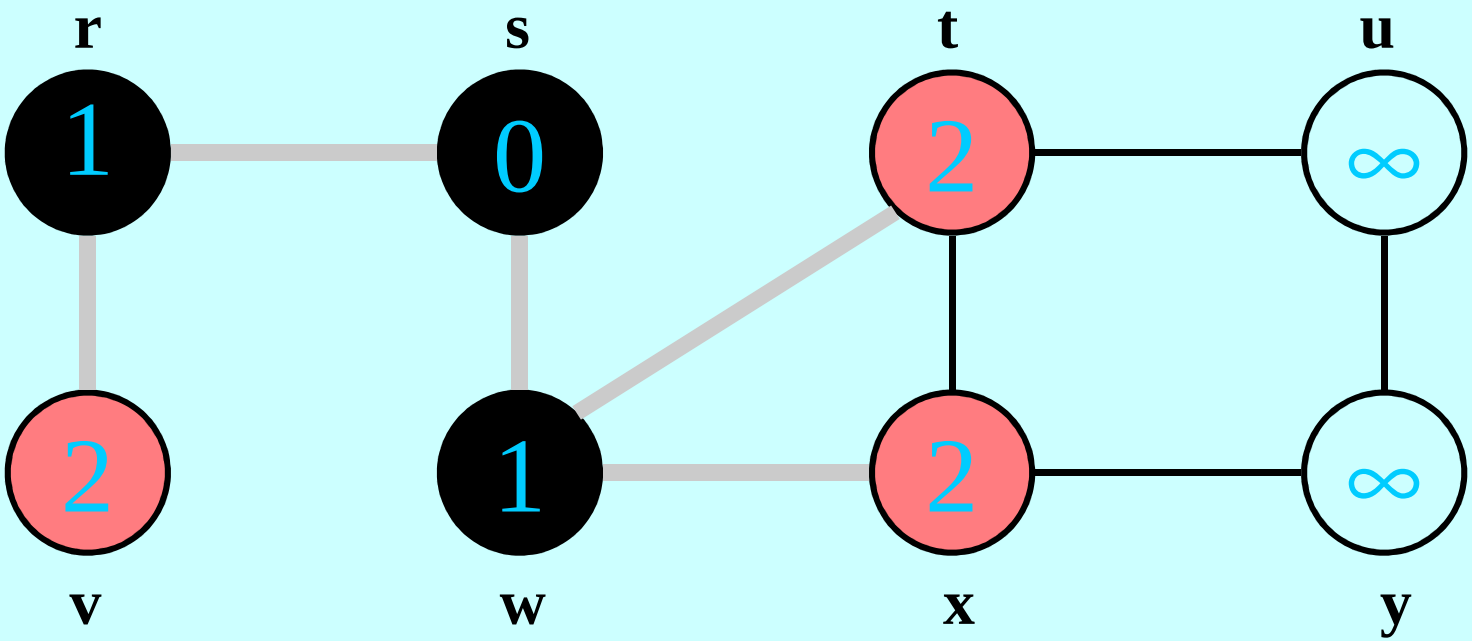
Q: | w | r |

Breadth-First Search: Example



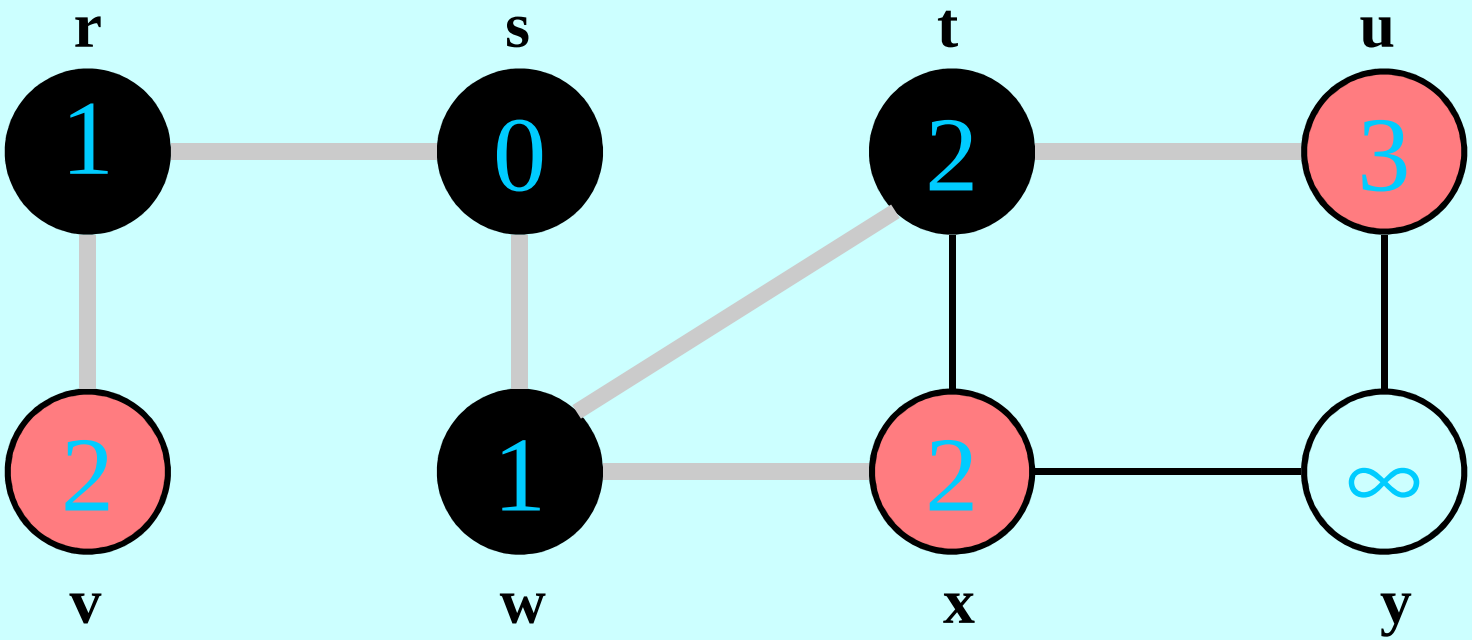
Q: | r | t | x |

Breadth-First Search: Example



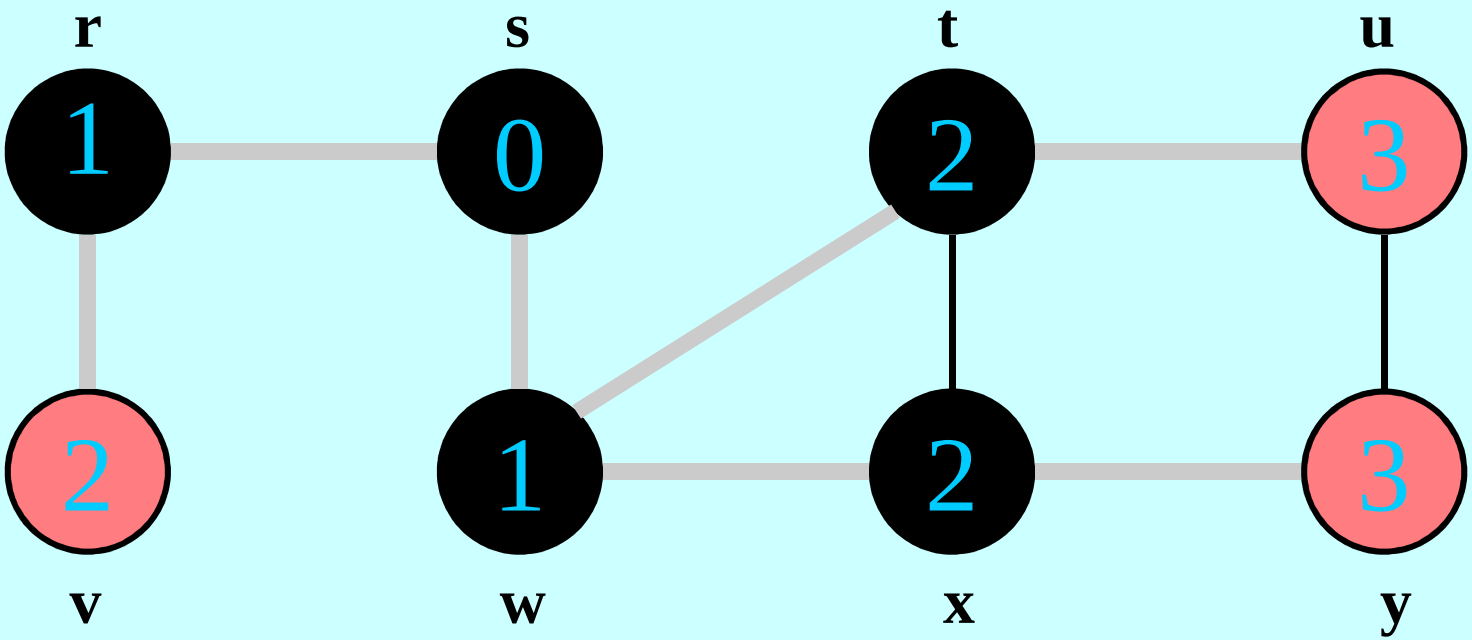
Q: | t | x | v |

Breadth-First Search: Example



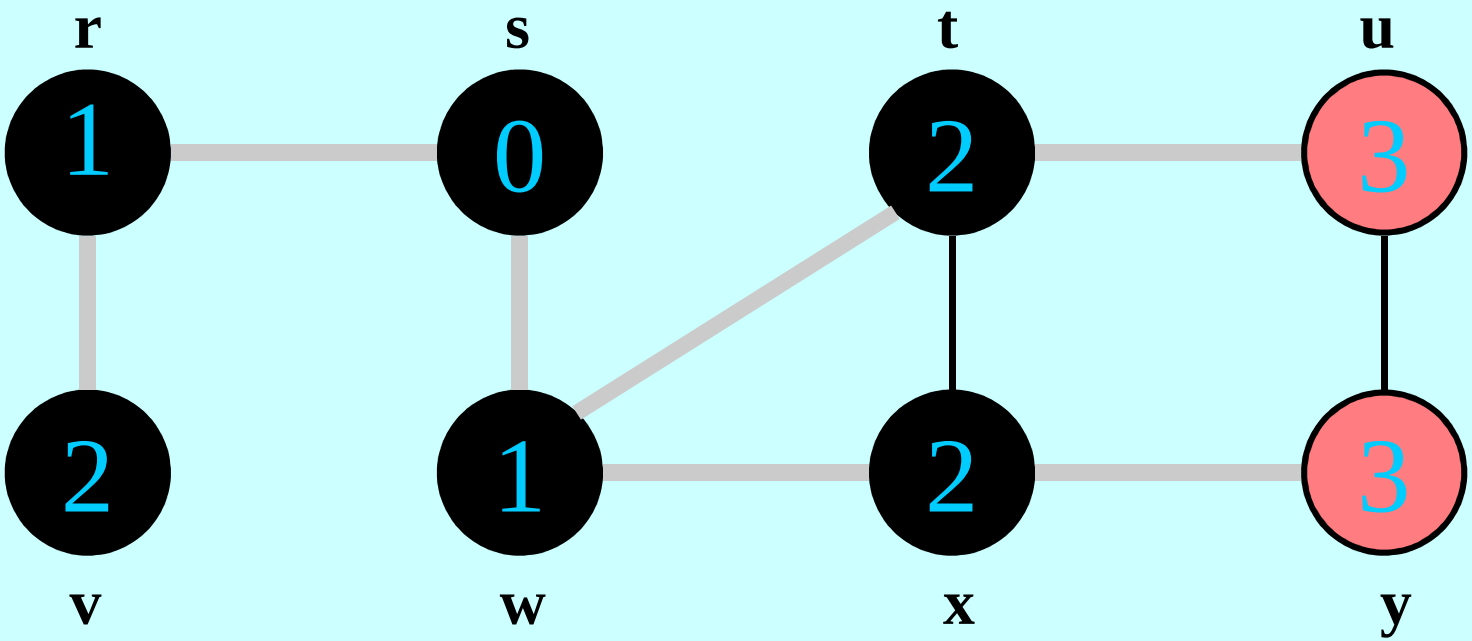
Q: | x | v | u |

Breadth-First Search: Example



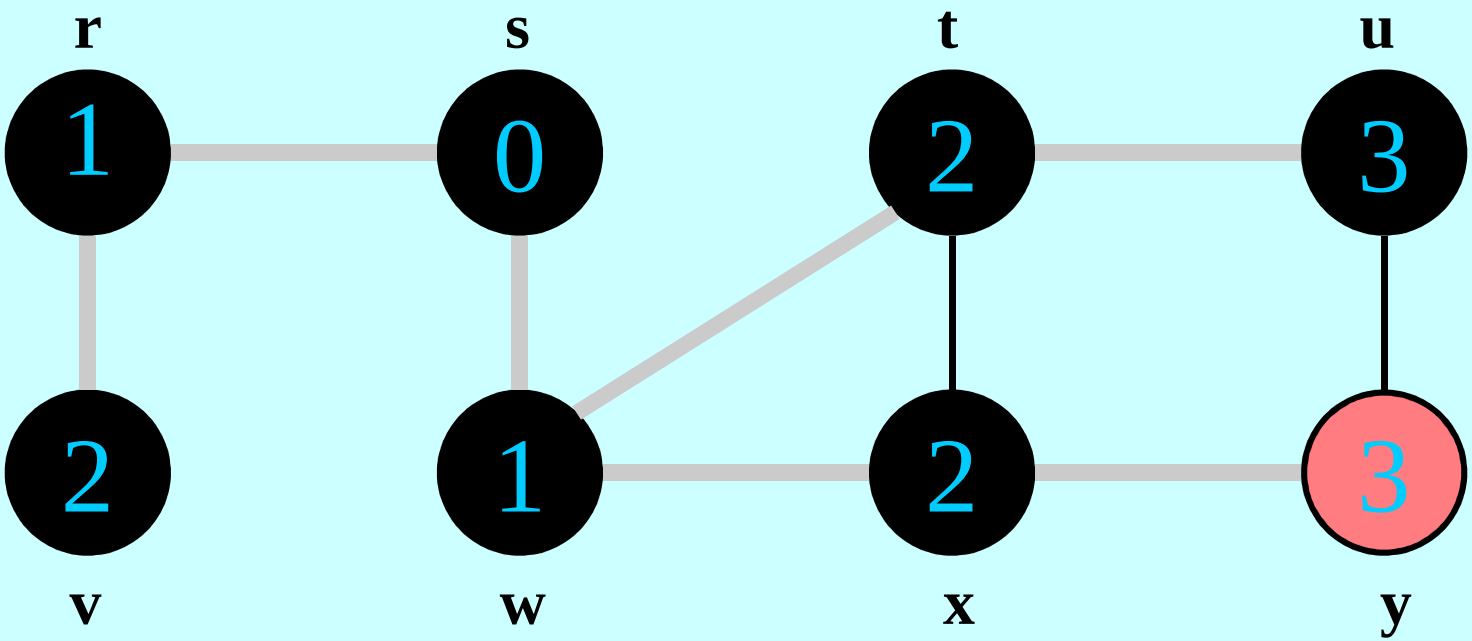
Q: | v | u | y |

Breadth-First Search: Example



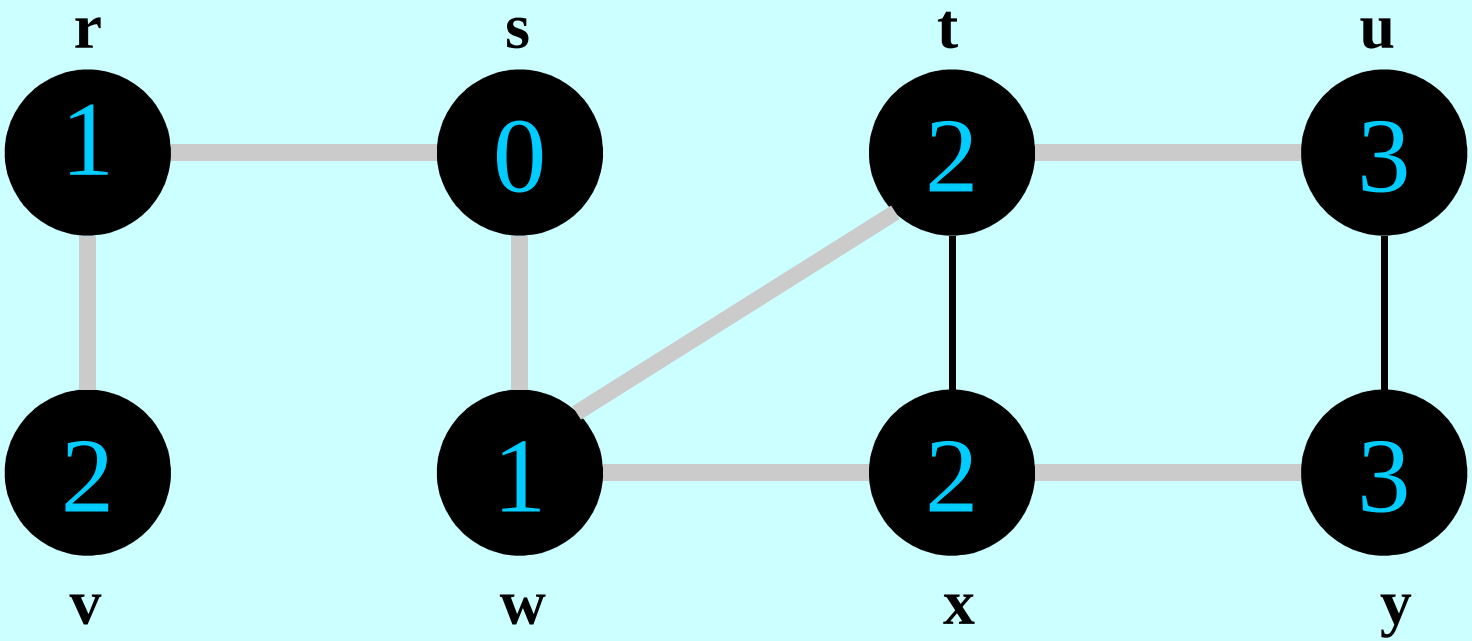
Q: | u | y |

Breadth-First Search: Example



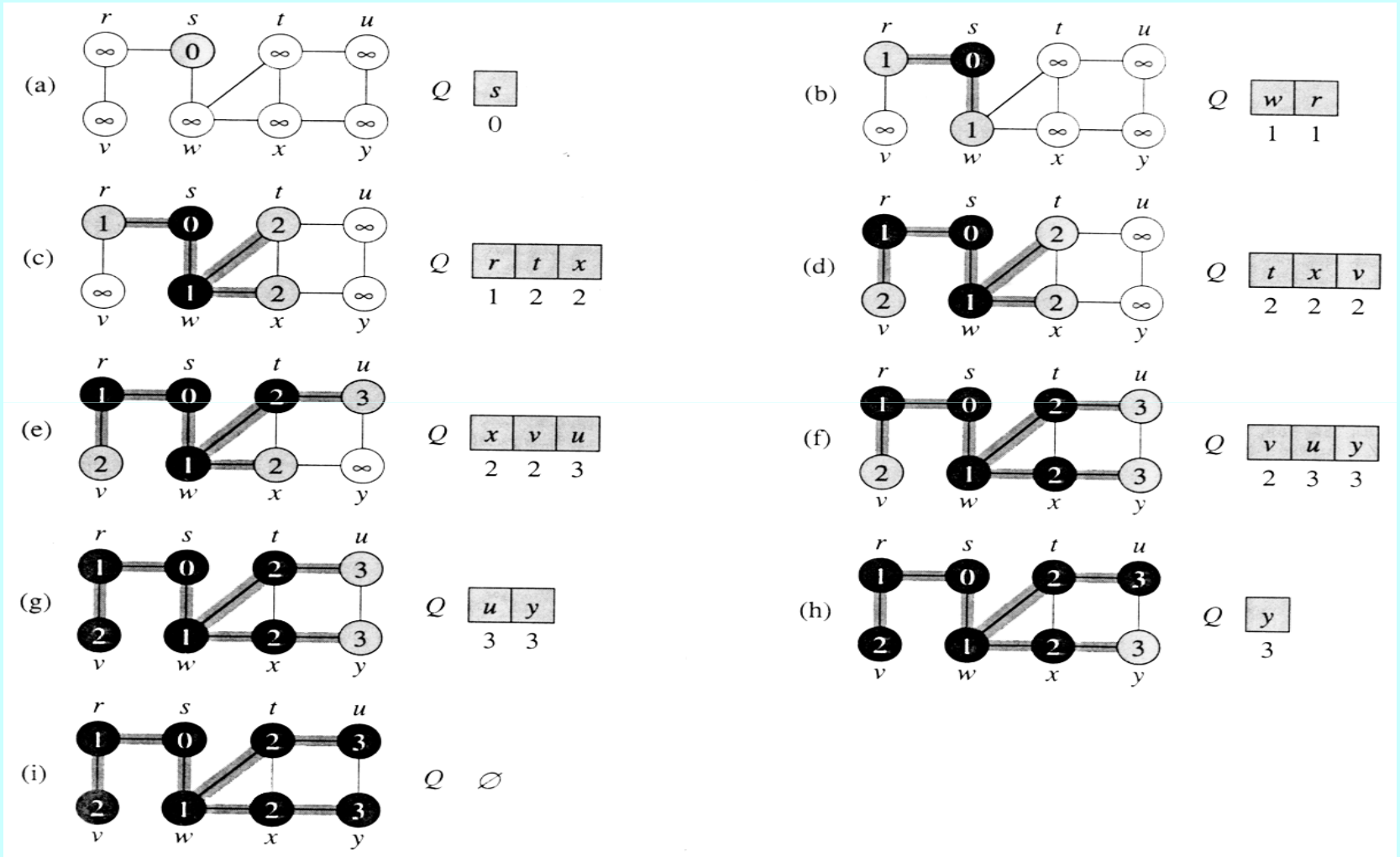
Q: y

Breadth-First Search: Example



Q: Ø

Operations of BFS on a Graph



BFS: The Code Again

```
BFS(G, s) {
    initialize vertices;
    Q = {s};
    while (Q not empty) {
        u = RemoveTop(Q);
        for each v ∈ u->adj {
            if (v->color == WHITE)
                v->color = GREY;
                v->d = u->d + 1;
                v->p = u;
                Enqueue(Q, v);
        }
        u->color = BLACK;
    }
}
```

Touch every vertex: O(V)

u = every vertex, but only once (Why?)

So v = every vertex that appears in some other vert's adjacency list

What will be the running time?

Total running time: O(V+E)

BFS: The Code Again

```
BFS(G, s) {
    initialize vertices;
    Q = {s};
    while (Q not empty) {
        u = RemoveTop(Q);
        for each v ∈ u->adj {
            if (v->color == WHITE)
                v->color = GREY;
                v->d = u->d + 1;
                v->p = u;
                Enqueue(Q, v);
        }
        u->color = BLACK;
    }
}
```

What will be the storage cost in addition to storing the tree?

Total space used:
O(max(degree(v))) = O(E)

Analysis of BFS

- Initialization takes $O(V)$.
- Loop Traversal
  - After initialization, each vertex is enqueued and dequeued at most once, and each operation takes $O(1)$. So, total time for queuing is $O(V)$.
  - The adjacency list of each vertex is scanned at most once. The sum of lengths of all adjacency lists is $\Theta(E)$.
- Summing up over all vertices
  - ❥ total running time of BFS is $O(V + E)$, linear in the size of the adjacency list representation of the graph.

Breadth-First Search: Properties

- BFS calculates the *shortest-path distance* to the source node
  - Shortest-path distance $\delta(s,v)$ = minimum number of edges from s to v, or ∞ if v not reachable from s
  - Proof given in the book (p. 472-5)
- BFS builds *breadth-first tree*, in which paths to root represent shortest paths in G
  - Thus can use BFS to calculate shortest path from one vertex to another in O(V+E) time

Shortest Paths

Definitions needed to prove BFS works:

- ***Shortest-Path distance*** $\delta(s, v)$ **from** s **to** v **is the minimum number of edges in any path from vertex** s **to vertex** v**, or else** ∞ **if there is no path from** s **to** v**.**

- **A path of length** $\delta(s, v)$ **from** s **to** v **is said to be a** ***shortest path*** **from** s **to** v**.**

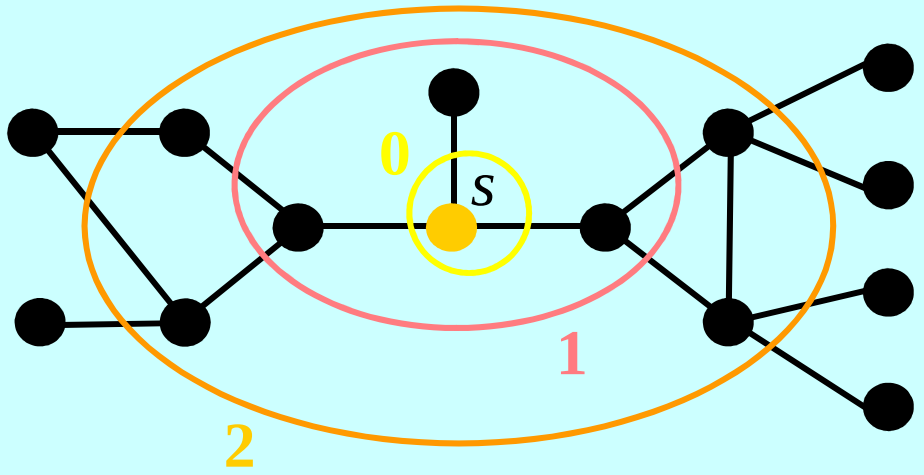
Lemmas

- 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 $(u, v) \in E$, $\delta(s, v) \leq \delta(s, u) + 1$.
  - *Proof*: If u is reachable from s, so is v; so it must take no more than one more step to reach v. If u is not reachable, $\delta(s, u) = \infty$. In both cases, the inequality holds.

- 22.2. 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 $v \in V$, the value $d[v]$ computed by BFS satisfies $d[v] \geq \delta(s, v)$.
  - The proof is by induction on the number of Enqueues.

Lemma 22.3

- Suppose that during the execution of BFS on a graph G, the queue Q contains vertices $(v_1, \ldots, v_r)$, 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 $i = 1, 2, \ldots, r - 1$.
  - The proof is by induction on the number of queue operations.
  - Example:



Depth-First-Search (DFS)

- This is a classic *back-track algorithm*.
- Explore edges out of the most recently discovered vertex v.
- When all edges of v have been explored, backtrack to explore edges leaving the vertex from which v was discovered (its *predecessor*).
- Search as deeply as possible first.
- Whenever a vertex v is discovered during a scan of the adjacency list of an already discovered vertex u, DFS records this event by setting predecessor $\pi[v]$ to u.

Depth-First Trees

- The coloring scheme is the same as for BFS. The predecessor subgraph of DFS is $G_\pi = (V, E_\pi)$ where $E_\pi = \{(\pi[v], v) : v \in V \text{ and } \pi[v] \neq NIL\}$. The predecessor subgraph G_π forms a *depth-first forest* composed of several *depth-first trees*. The edges in E_π are called *tree edges*.
- Each vertex u has two *timestamps*: $d[u]$ records when u is first discovered (grayed) and $f[u]$ records when the search finishes (blackens). For every vertex u, $d[u] < f[u]$.

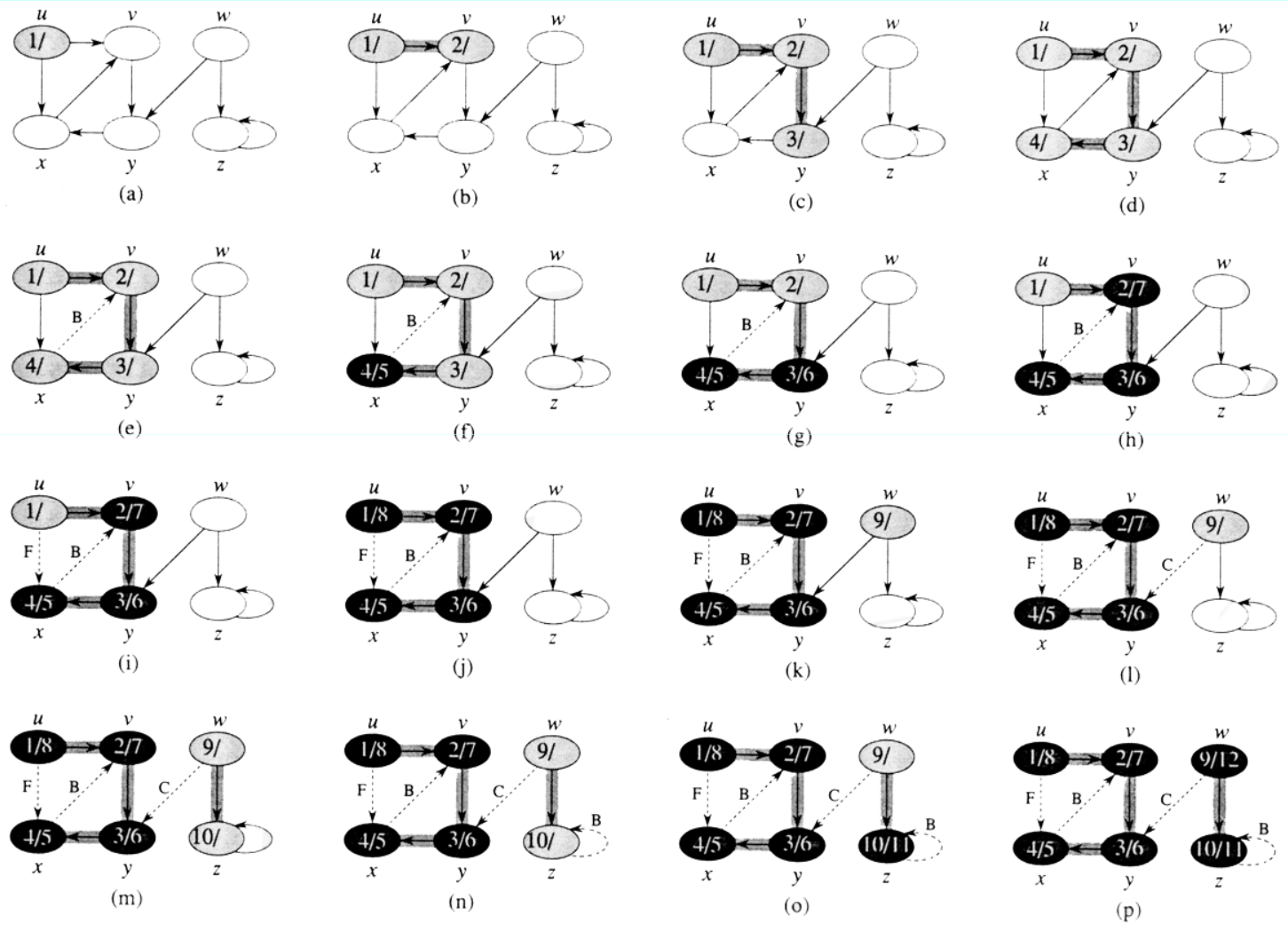
DFS(G)

1. for each vertex $u \in V[G]$
2. do $color[u] \leftarrow$ White
3. $\pi[u] \leftarrow$ Nil
4. $time \leftarrow 0$
5. for each vertex $u \in V[G]$
6. do if $color[v]$ = White
7. then DFS-Visit(v)

DFS-Visit(u)

1. $color[u] \leftarrow$ GRAY
   - White vertex u has been discovered
2. $time \leftarrow time + 1$
3. $d[u] \leftarrow time$
4. for each vertex $v \in Adj[u]$ — Explore edge (u, v)
5. do if $color[v]$ = WHITE
6. then $\pi[v] \leftarrow u$
7. DFS-Visit(v)
8. $color[u] \leftarrow$ BLACK — Blacken u; it is finished.
9. $time \leftarrow time + 1$
10. $f[u] \leftarrow time$

Operations of DFS



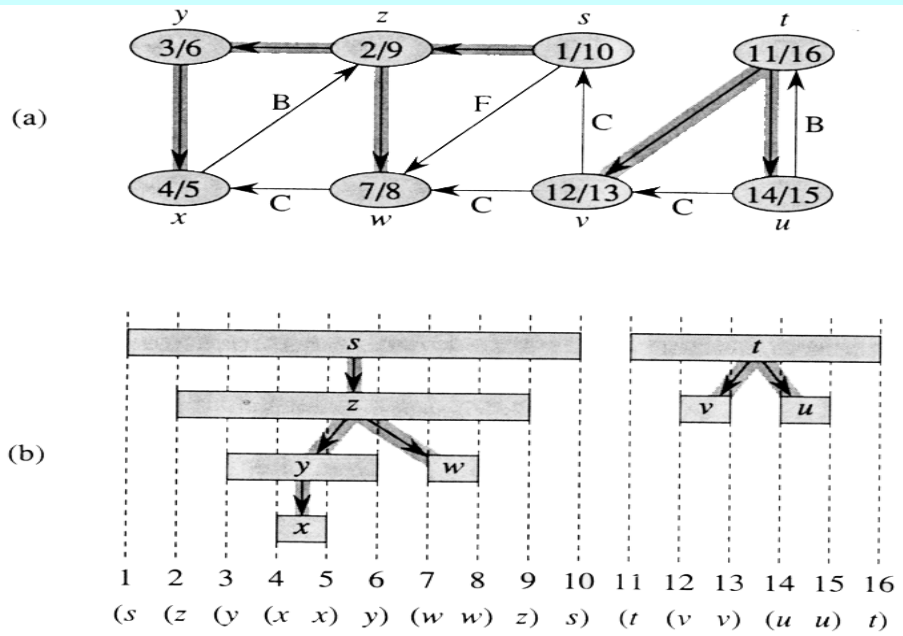
Analysis of DFS

- Loops on lines 1-2 and 5-7 take $\Theta(V)$ time, excluding the time to execute DFS-Visit.
- DFS-Visit is called once for each white vertex $v \in V$, when it is painted gray the first time. Lines 3-6 of DFS-Visit is executed $|\text{Adj}[v]|$ times. The total cost of executing DFS-Visit is $\sum_{v \in V} |\text{Adj}[v]| = \Theta(E)$.
- The total running time of DFS is $\Theta(V + E)$.

Properties of DFS

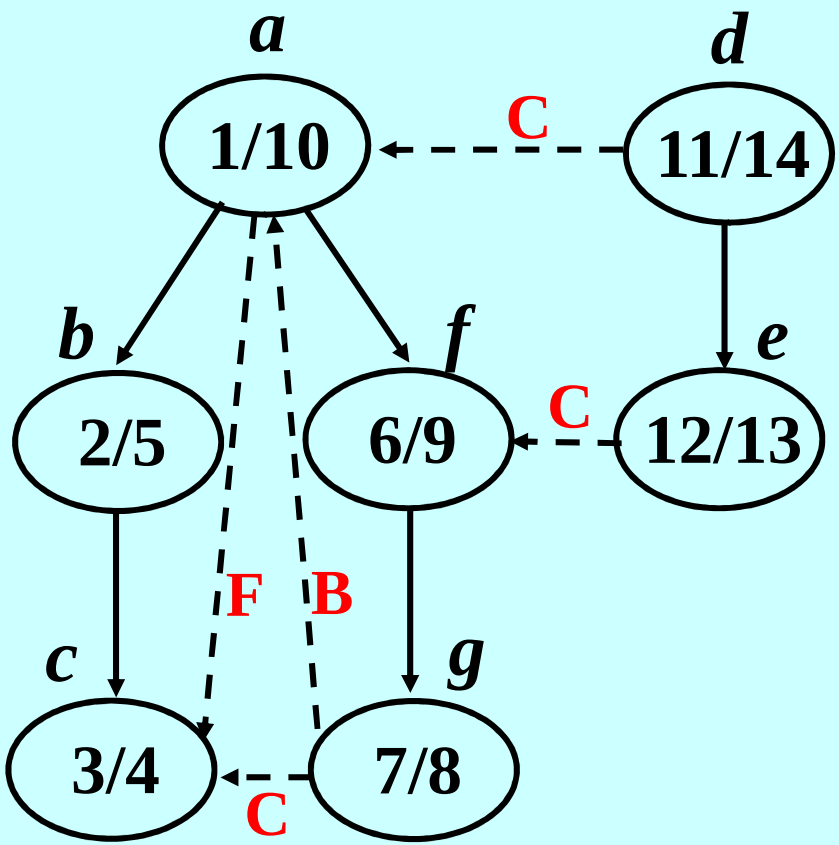
- Predecessor subgraph G_π forms a forest of trees (the structure of a depth-first tree mirrors the structure of DFS-Visit).

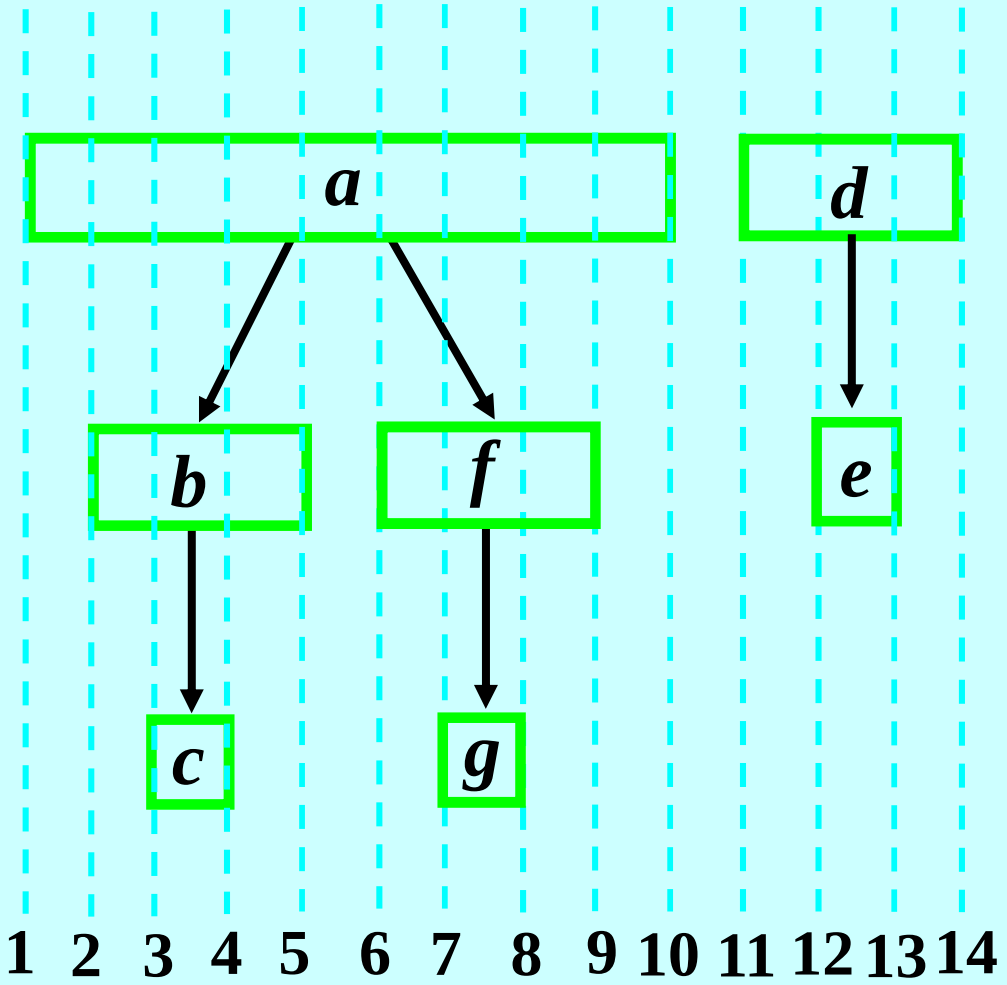
- The discovery and finishing time have *parenthesis structure*, i.e. the parentheses are properly nested. (See the figures and next theorem)



[redacted] Properties of depth-first search. **(a)** The result of a depth-first search of a directed graph. Vertices are timestamped and edge types are indicated as in Figure 2[redacted].4. **(b)** Intervals for the discovery time and finishing time of each vertex correspond to the parenthesization shown. Each rectangle spans the interval given by the discovery and finishing times of the corresponding vertex. Tree edges are shown. If two intervals overlap, then one is nested within the other, and the vertex corresponding to the smaller interval is a descendant of the vertex corresponding to the larger. **(c)** The graph of part (a) redrawn with all tree and forward edges going down within a depth-first tree and all back edges going up from a descendant to an ancestor.

DFS & Parenthesis Structure





Lecture Summary

- Analysed basic graph search algorithms
  - DFS, BFS
- Properties
  - Complexity, correctness