CS 332: Algorithms

Graph Algorithms

https://www.cs.virginia.edu/~luebke/cs332.fall00/lecture18.ppt

Review: Graphs

- A graph $G = (V, E)$
  - V = set of vertices, E = set of edges
  - *Dense* graph: $|E| \approx |V|^2$; *Sparse* graph: $|E| \approx |V|$
  - *Undirected graph:*
    - Edge (u,v) = edge (v,u)
    - No self-loops
  - *Directed* graph:
    - Edge (u,v) goes from vertex u to vertex v, notated $u \rightarrow v$
  - A *weighted graph* associates weights with either the edges or the vertices

Review: Representing Graphs

- Assume V = {1, 2, …, *n*}
- An *adjacency matrix* represents the graph as a *n* x *n* matrix A:
  - A[*i*, *j*] = 1 if edge (*i*, *j*) $\in$ E (or weight of edge)
    = 0 if edge (*i*, *j*) $\notin$ E
  - Storage requirements: $O(V^2)$
    - A dense representation
  - But, can be very efficient for small graphs
    - Especially if store just one bit/edge
    - Undirected graph: only need one diagonal of matrix

Universal Sink

- Show how to determine whether a directed graph G contains a universal sink - a vertex with in-degree (V-1) (V is the number of vertices) and out-degree 0, given an adjacency matrix for G. Can be done in O(V)

Review: Breadth-First Search

- “Explore” a graph, turning it into a tree
  - One vertex at a time
  - Expand frontier of explored vertices across the *breadth* of the frontier
- Builds a tree over the graph
  - Pick a *source vertex* to be the root
  - Find (“discover”) its children, then their children, etc.

Review: Breadth-First Search

- Again will associate vertex “colors” to guide the algorithm
  - White vertices have not been discovered
    - All vertices start out white
  - Grey 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 grey vertices

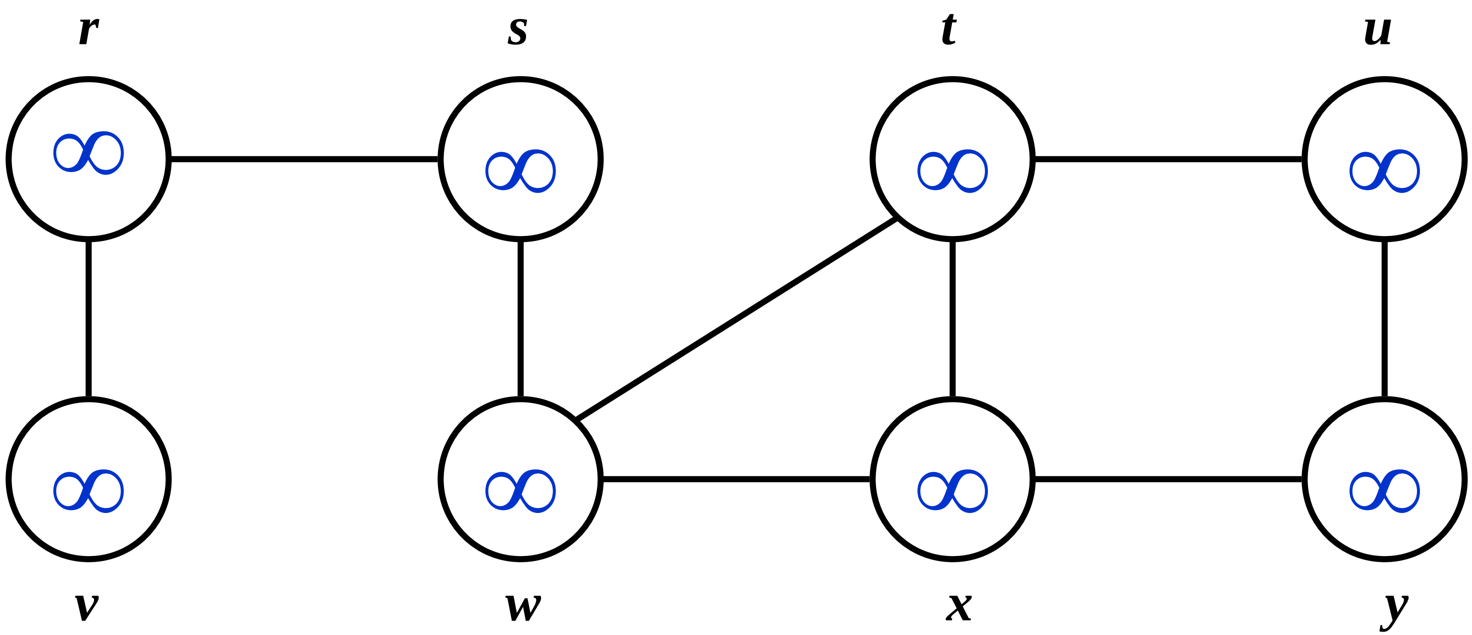
Review: Breadth-First Search

```
BFS(G, s) {
    initialize vertices;
    Q = {s};        // Q is a queue (duh); initialize to 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;
    }
}
```

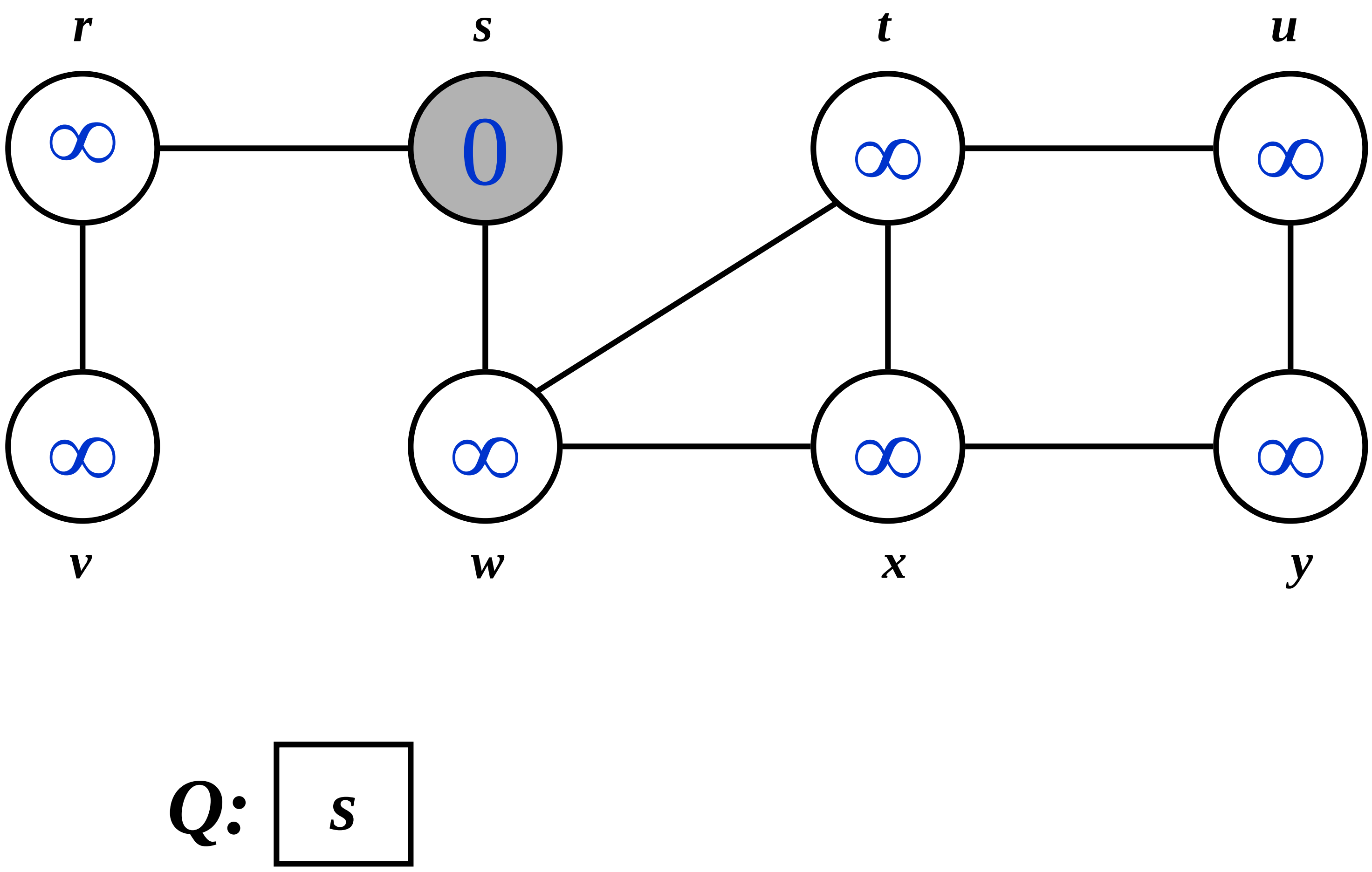
`v->d` *represents level of v*

`v->p` *represents the parent of v*

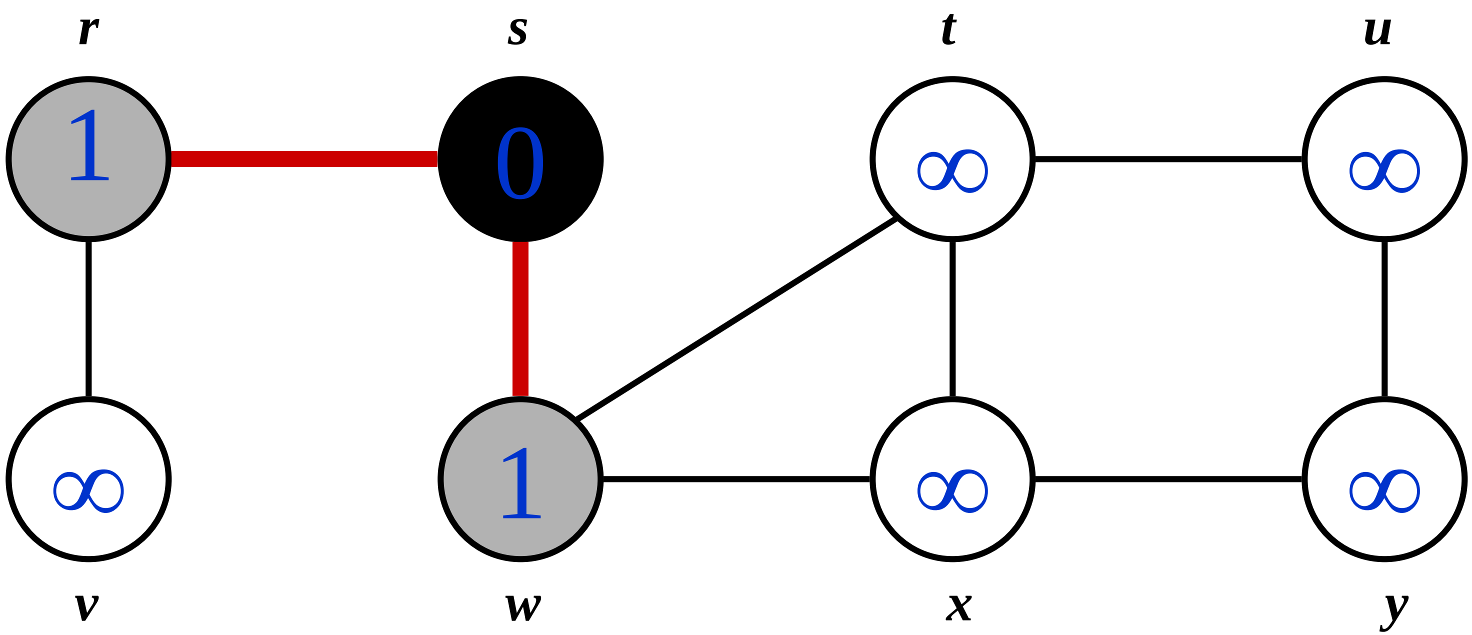
Breadth-First Search: Example

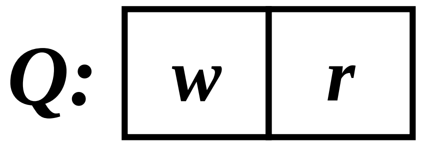


Breadth-First Search: Example

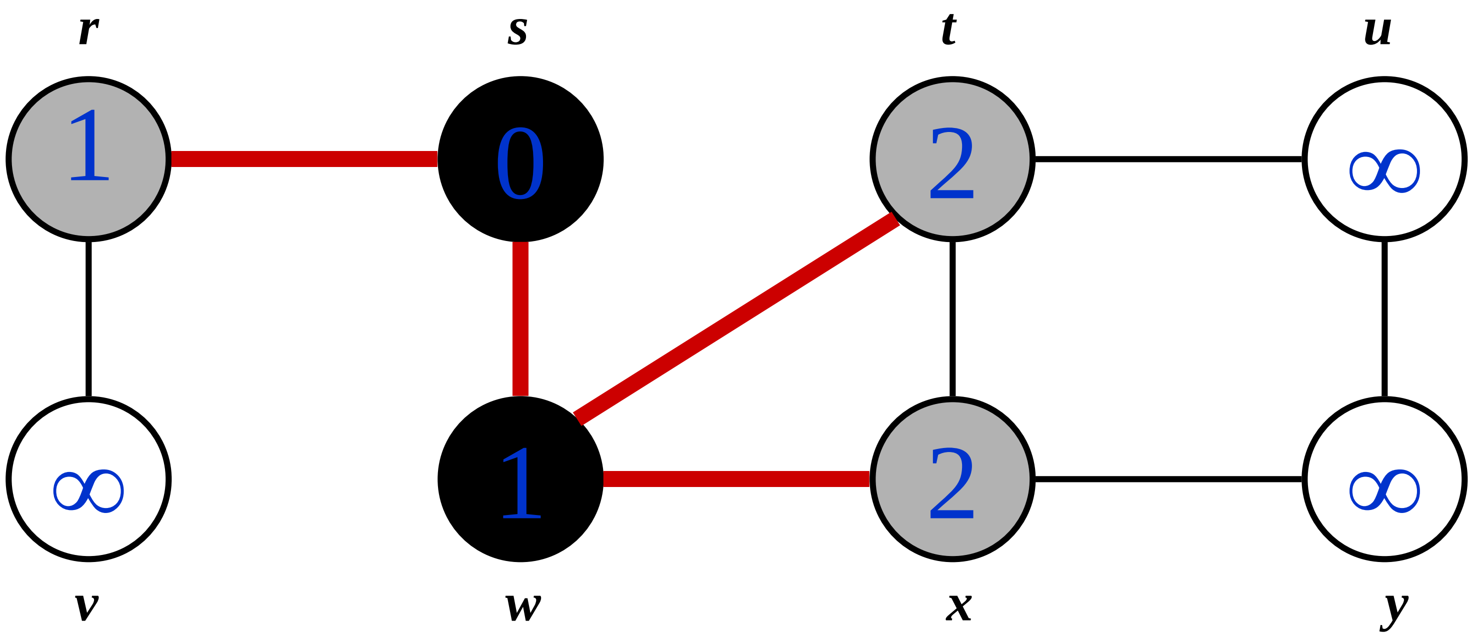


Breadth-First Search: Example



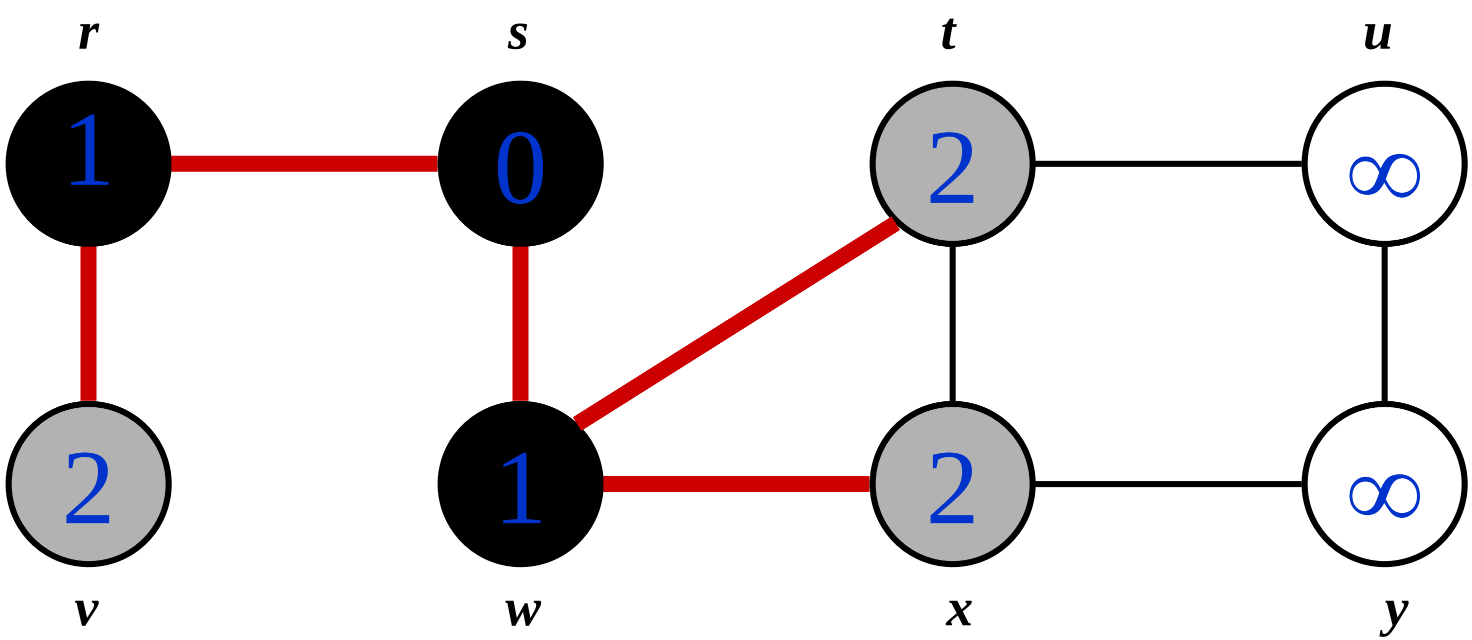


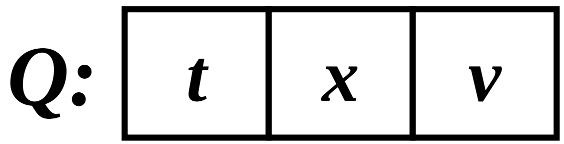
Breadth-First Search: Example



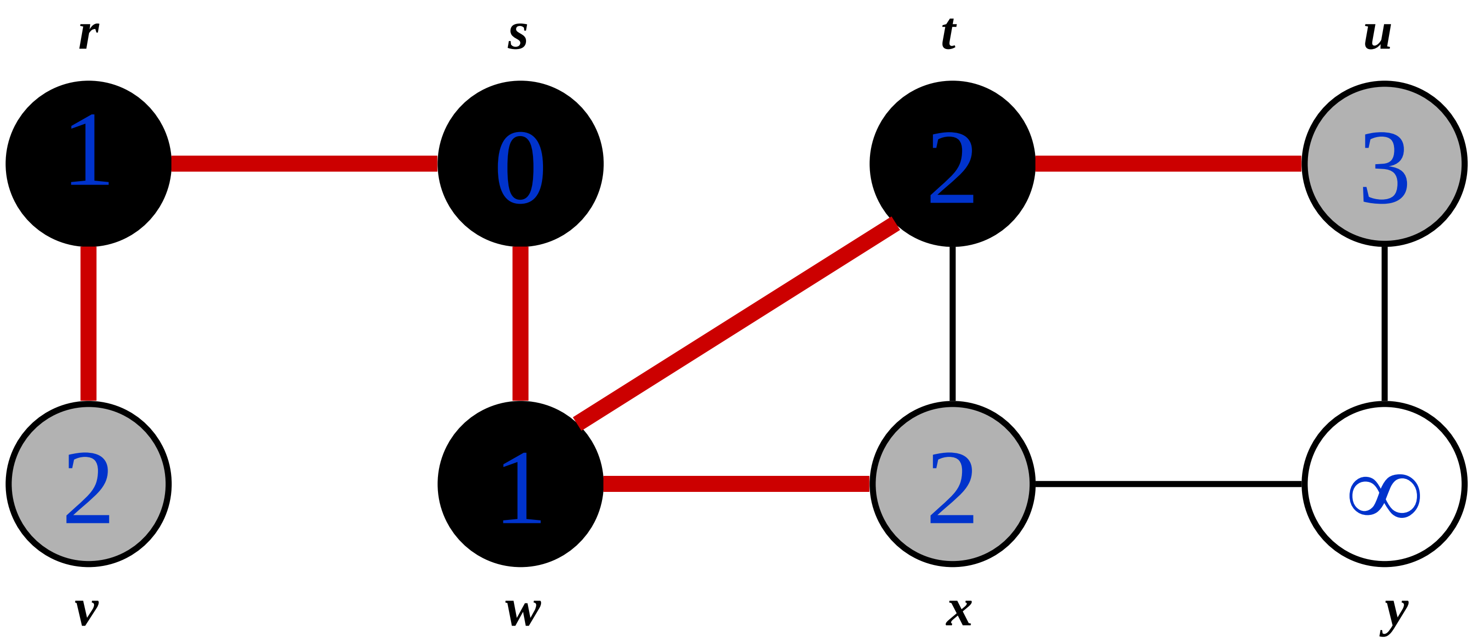
Q: | r | t | x |

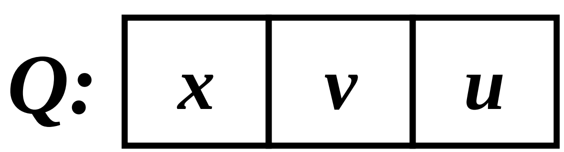
Breadth-First Search: Example



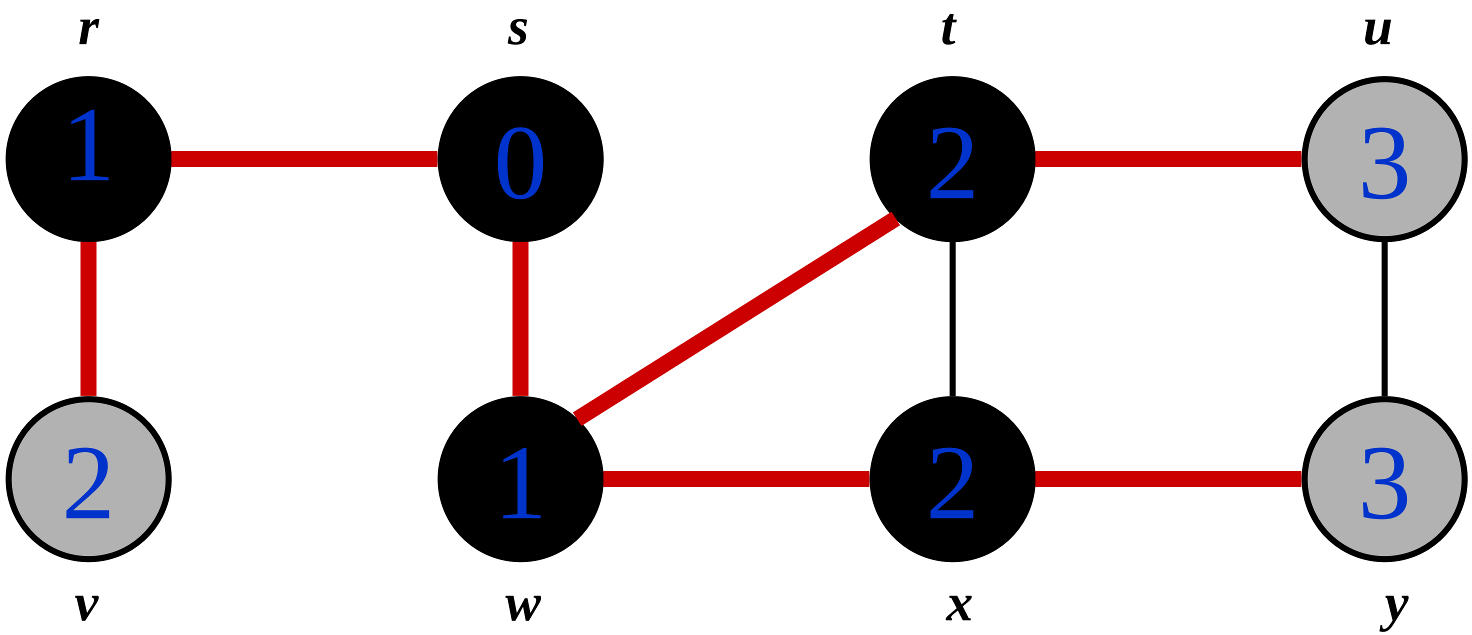


Breadth-First Search: Example



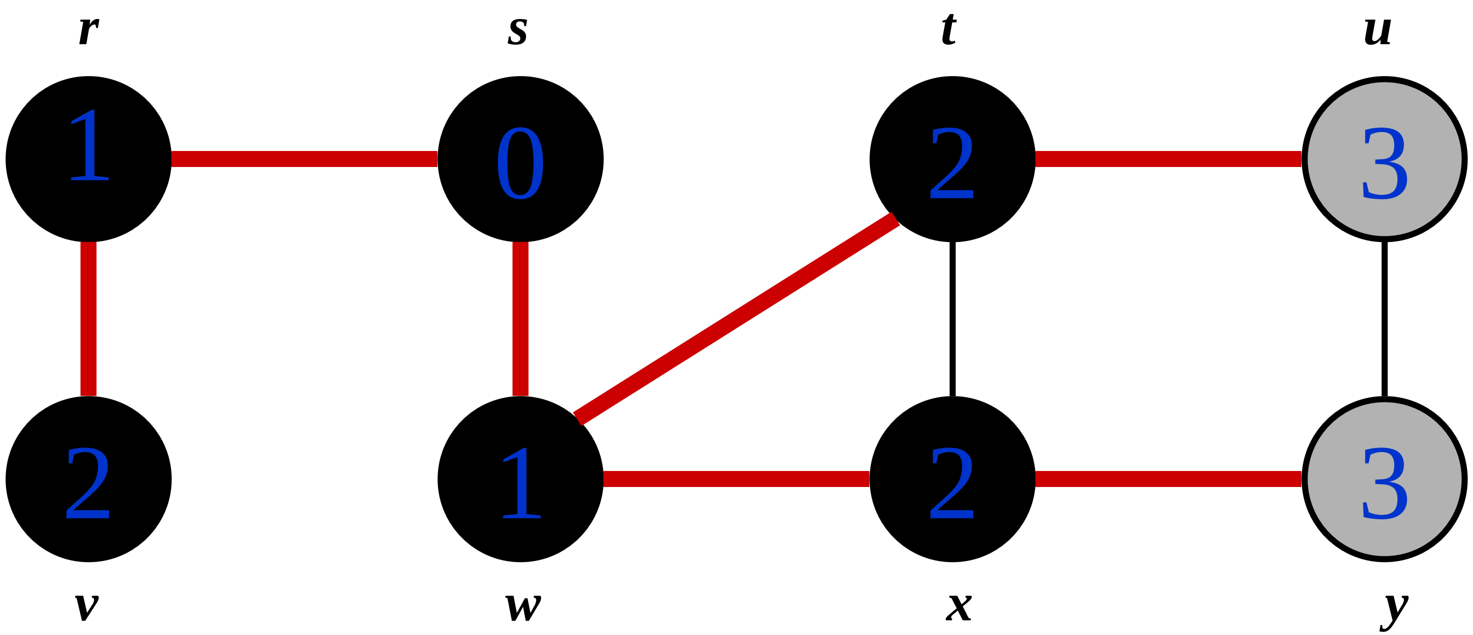


Breadth-First Search: Example



Q:			
	v	*u*	*y*

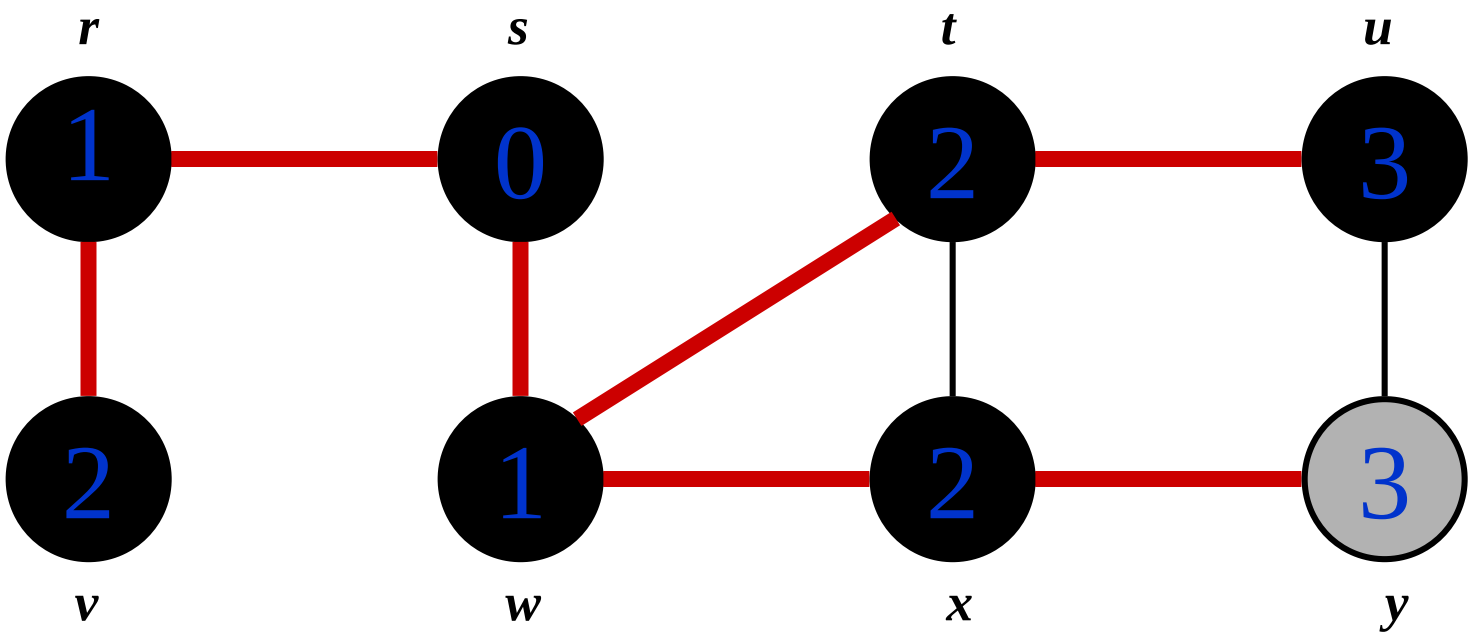
Breadth-First Search: Example



Q:

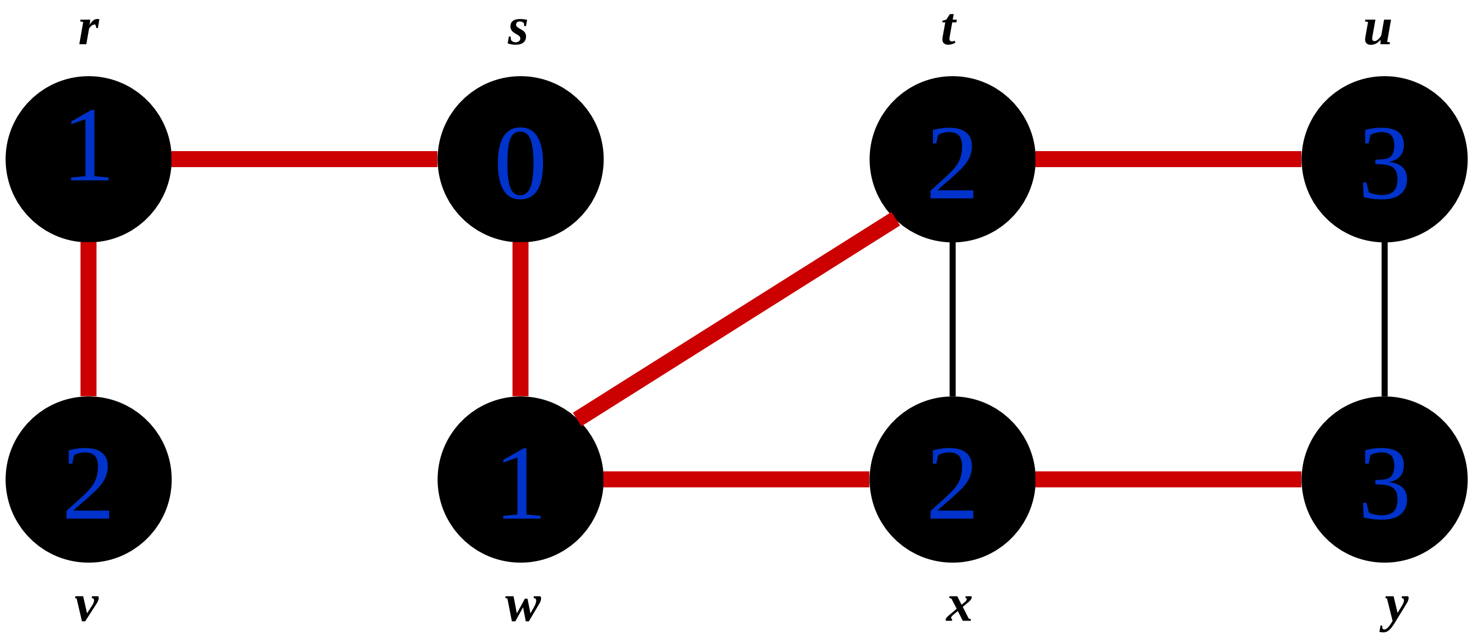
u	*y*

Breadth-First Search: Example



Q: | *y* |

Breadth-First Search: Example



Q: Ø

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)

Breadth-First Search: Properties

- BFS calculates the *shortest-path distance* to the source node
- Since BFS gives a path from root to node, d[v]>=sd[v] (cant be shorter than shortest path)
- To prove that d[v]!> sd[v]
- D[v] is distance reported by bfs, sd[v] is shortest possible distance to v from s

Proof sketch

- Let v be node closest to s that has d[v]!=sd[v].
- Now d[v] cant be less than sd[v] as sd[v] is the shortest possible distance. Hence to prove d[v]!>sd[v].
- Consider the actual shortest path. Let u be vertex just before v on shortest path from s to v which means sd[v]=sd[u]+1
- Lets look at what could have happened during the bfs when u was being explored. Note u is grey.
  -either v was white or black or grey. All show contradiction
  If v was white: then d[v]=d[u]+1

  If v was black: then v was enqueued before u so d[v] <=d[u]

  If v was grey: so v was discovered through say w. Hence d[w] was enqueued before u and hence d[w]<=d[u]. Hence, d[w]+1<=d[u]+1. Hence d[v]=d[w]+1<=d[u]+1.