Graphs, Part 1

Ira A. Fulton Schools of Engineering
Arizona State University

The Seven Bridge Problem

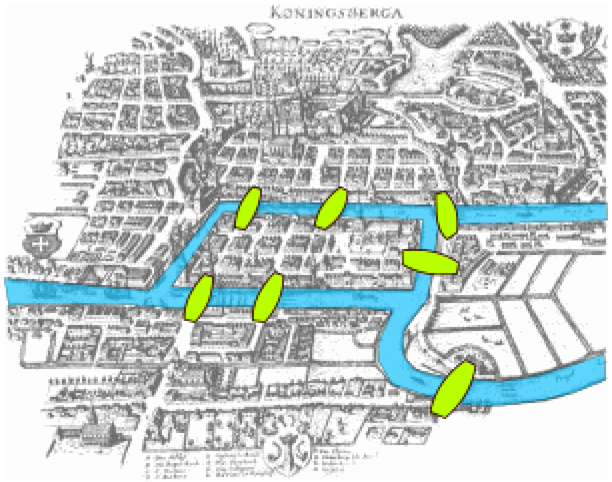
- The city of Königsberg was set on both sides of the Pregel River, and includes two large islands—Kneiphof and Lomse—which were connected to each other, or to the two mainland portions of the city, by seven bridges.
- The problem was to devise a walk through the city that would cross each of those bridges once and only once.

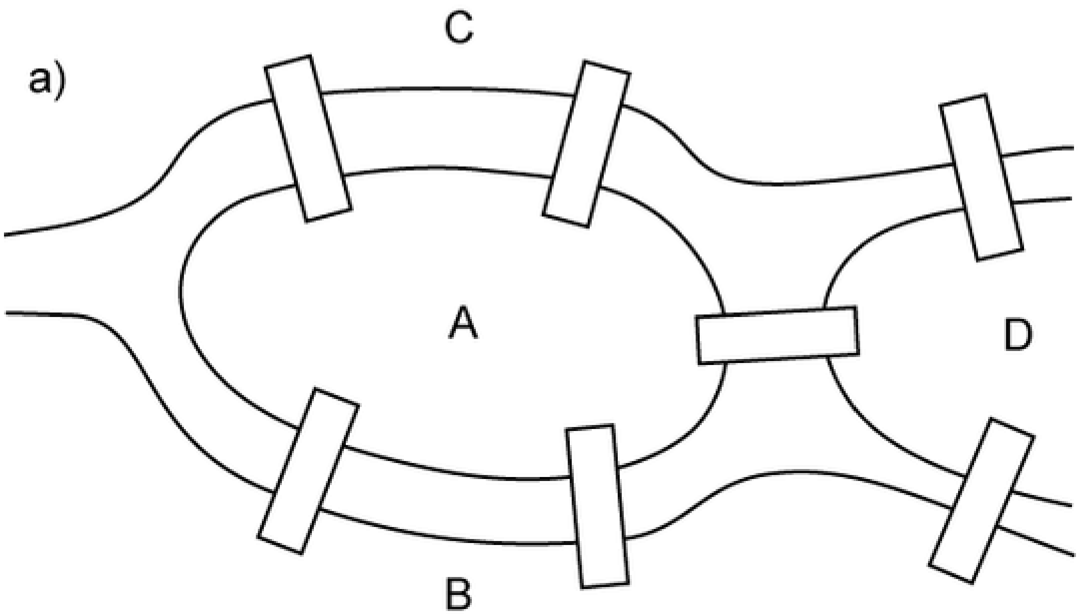
Euler's View of the Problem

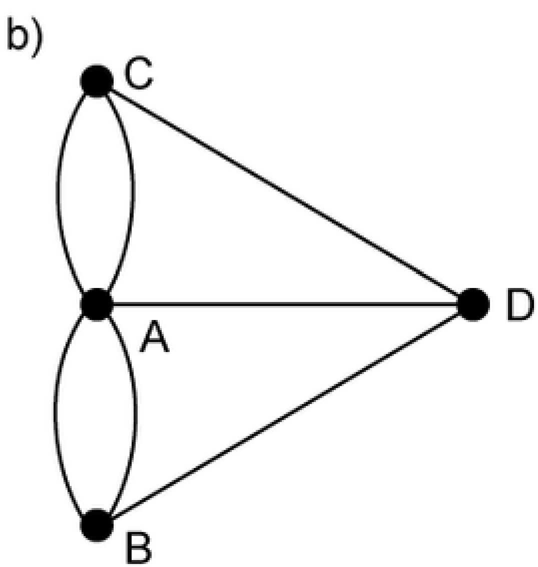
- We can abstract the problem using the concept of graphs.
- Each piece of land (island) is denoted by a vertex.
- There are 4 vertices: A, B, C, D.
- Each bridge connecting two pieces of land (island) is denoted by an edge.
- There are 7 edges.
- The number of edges incident with vertex v is called the degree of vertex v.

Euler's Observation

- **Euler's solution**







Euler's Observation

- If the tour starts and ends at the same vertex v, the degree of v must be an even number.
- If the tour passes vertex v but does not start or end at v, the degree of v must be an even number.
- Any tour that goes through all 7 edges will touch each of the 4 vertices at least once.
- The degree for each of the 4 vertices is an odd number.
- The tour cannot start and end at the same vertex.
- There are more than 2 vertices whose degree is an odd number.

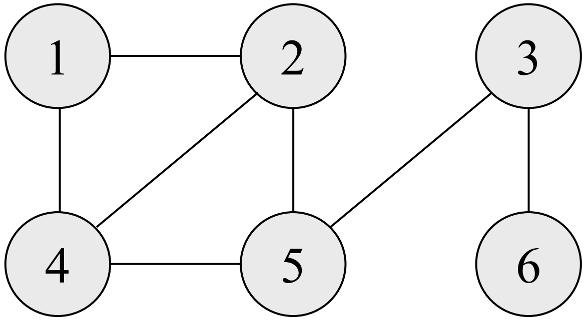
Euler’s Observation

- Conclusion: There is no Euler tour for the seven-bridge problem of Königsberg

- This is the origin of Graph Theory

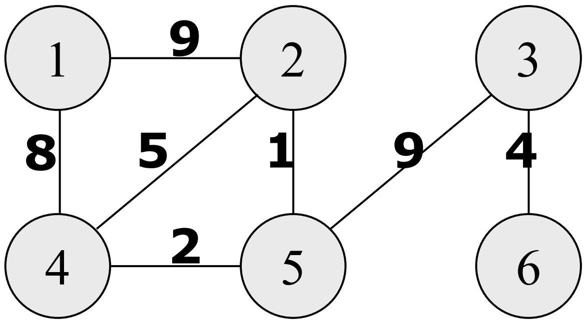
Undirected Graph

- G = (V, E), where V is the set of vertices, and E is the set of undirected edges.
- Each edge is an unordered pair of vertices.
- In the graph below, V={1, 2, 3, 4, 5, 6}, E={(1,2), (1, 4), (2, 4), (2, 5), (3, 5), (3, 6), (4, 5)}.
- Note that (2, 1) and (1, 2) denote the same edge.



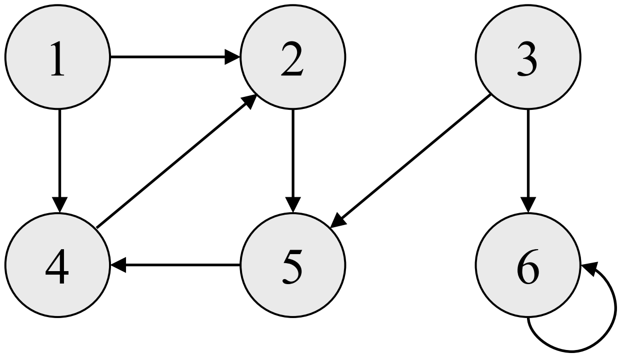
Weighted Undirected Graph

- G = (V, E, w), where V is the set of vertices, E is the set of undirected edges, and w: E→R is an edge weight function.
- In the example below, w(1, 2)=9, w(3, 6)=4.
- Edge weight can represent cost, distance, bandwidth, etc., depending on applications.



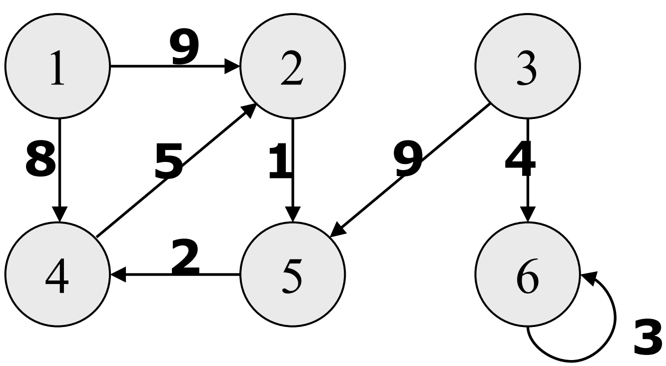
Directed Graph

- G = (V, E), where V is the set of vertices, and E is the set of directed edges.
- Each edge is an ordered pair of vertices.
- In the graph below, V={1, 2, 3, 4, 5, 6}, E={(1,2), (1, 4), (2, 5), (3, 5), (3, 6), (4, 2), (5, 4), (6, 6)}.



Weighted Directed Graph

- G = (V, E, w), where V is the set of vertices, E is the set of directed edges, and w: E→R is an edge weight function.
- In the example below, w(1, 2)=9, w(2, 5)=1.
- Edge weight can represent cost, distance, bandwidth, etc., depending on applications.



Summary

- Graphs and graph theory originated from the seven-bridge problem.
- We can classify graphs into directed graphs and undirected graphs.
- We can classify graphs into weighted graphs and unweighted graphs.

Ira A. Fulton Schools of Engineering

Arizona State University

Graphs, Part 2

Ira A. Fulton Schools of Engineering
Arizona State University

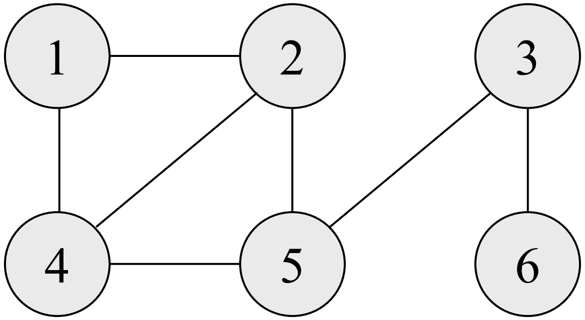
Graph Representations

- There are two main representations of graphs: Adjacency matrix and Adjacency lists
- You need to choose the right representation for your applications.

Adjacency Matrix for Unweighted Graphs

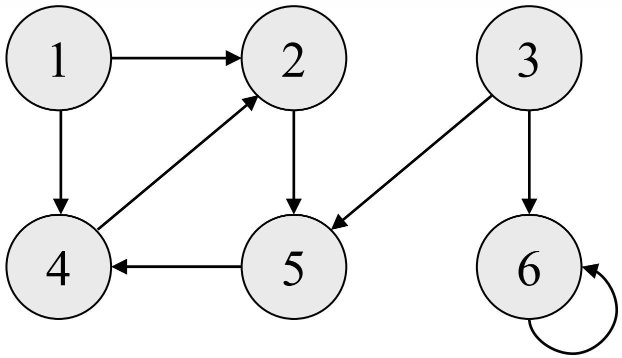
- Assume that $|V|=n$ and $|E|=m$. Adjacency matrix for unweighted graph $G=(V, E)$ is an n-by-n matrix $A=(a_{ij})$
- $a_{ij} = 1$ if $(i, j) \in E$, and $a_{ij} = 0$ otherwise
- Commonly we omit the 0s in the matrix

Adjacency Matrix for Undirected Graphs



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Adjacency Matrix for Directed Graphs



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

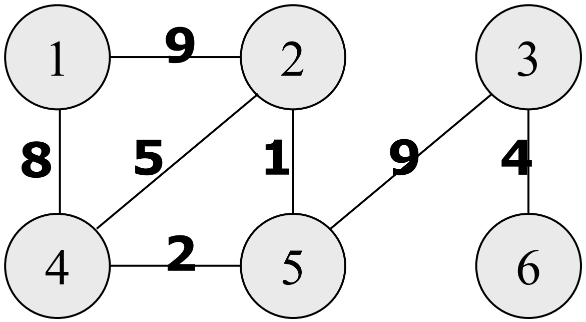
Adjacency Matrix for Weighted Graphs

- Assume that $|V|=n$ and $|E|=m$. Adjacency matrix for weighted graph $G=(V, E, w)$ is an n-by-n matrix $A=(a_{ij})$

- $a_{ij} = w(i, j)$ if $(i, j) \in E$, and a_{ij} = NIL (or ∞) o/w

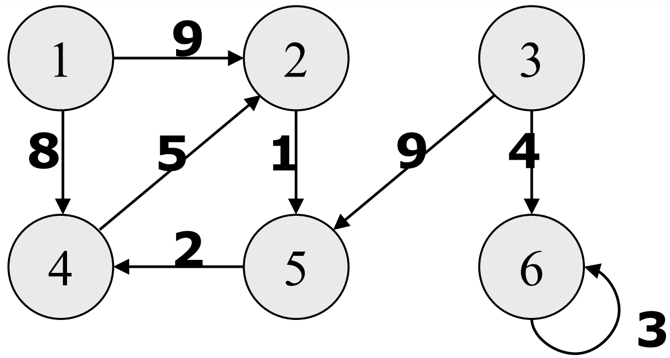
- Commonly we omit the NILs in the matrix

Adjacency Matrix for Weighted Graphs



$$A = \begin{pmatrix} \infty & 9 & \infty & 8 & \infty & \infty \\ 9 & \infty & \infty & 5 & 1 & \infty \\ \infty & \infty & \infty & \infty & 9 & 4 \\ 8 & 5 & \infty & \infty & 2 & \infty \\ \infty & 1 & 9 & 2 & \infty & \infty \\ \infty & \infty & 4 & \infty & \infty & \infty \end{pmatrix}$$

Adjacency Matrix for Weighted Graphs

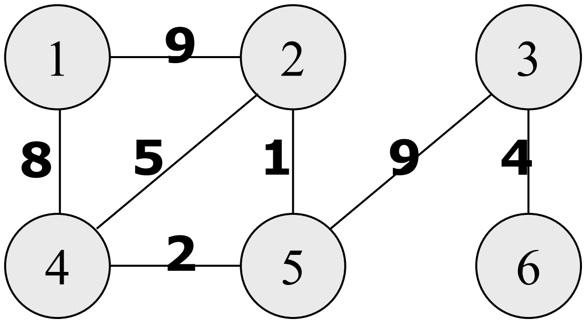


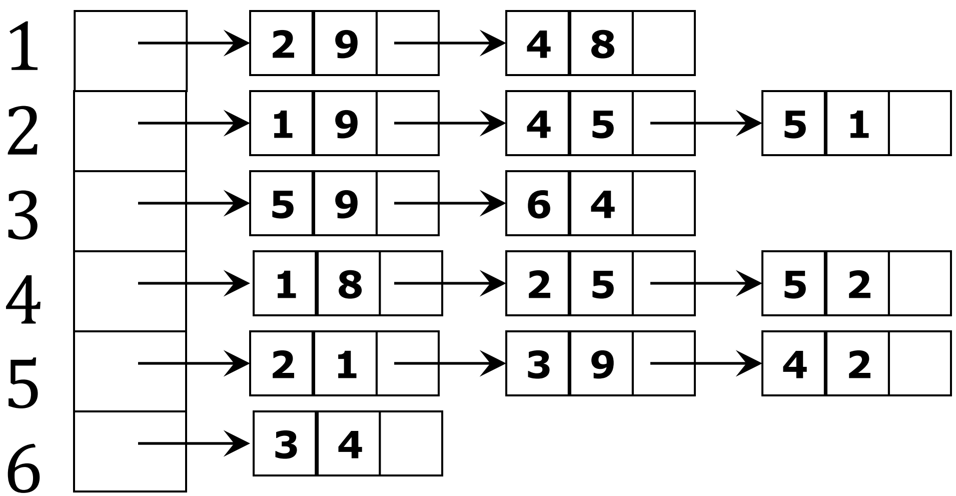
$$A = \begin{pmatrix} \infty & 9 & \infty & 8 & \infty & \infty \\ \infty & \infty & \infty & \infty & 1 & \infty \\ \infty & \infty & \infty & \infty & 9 & 4 \\ \infty & 5 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 3 \end{pmatrix}$$

Adjacency Lists for Weighted Graphs

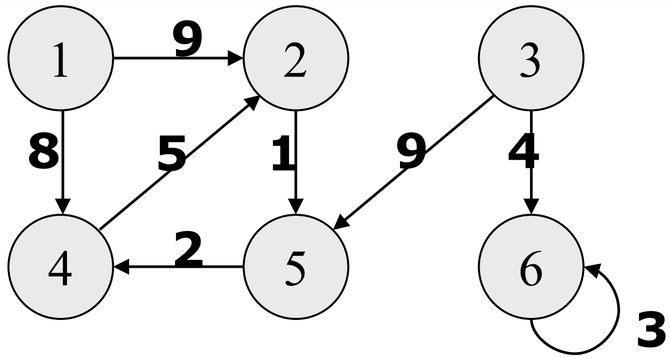
- Assume that |V|=n and |E|=m. Adjacency lists for weighted graph G=(V, E, w) is an array G.Adj[], where for each $u \in V$, G.Adj[u] (also by u.Adj) is a list that contains all vertices v such that $(u, v) \in E$.
- G.Adj[u] is a pointer (could be NULL).
- Each node on the list is a struct with the fields
  - vertex, which is the label of the vertex v
  - weight, which is the value of w(u, v)
  - next, which points to the next node on the list
  - there is no weight field for unweighted graphs

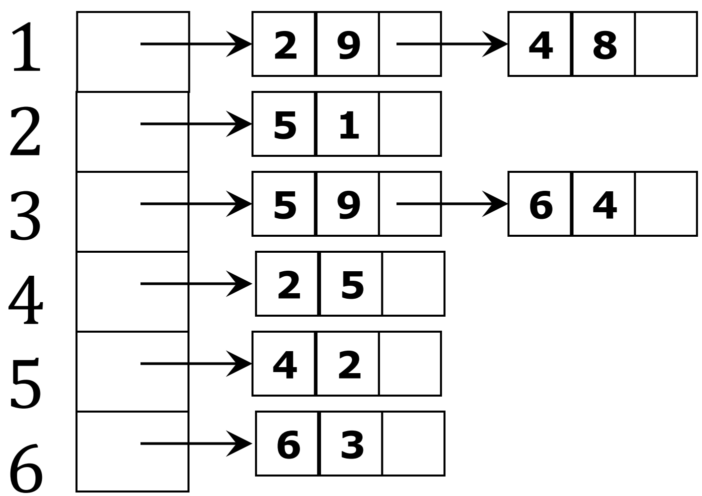
Adjacency Lists for Weighted Graphs





Adjacency Lists for Weighted Graphs





ASU®

Ira A. Fulton Schools of
Engineering

Arizona State University

Graphs, Part 3

Ira A. Fulton Schools of Engineering
Arizona State University

Breadth-First-Search

- One of the simplest algorithms for graph searching
- Very efficient
- Many applications
  - Prim’s minimum-spanning-tree algorithm
  - Dijkstra’s shortest-path algorithm

Preliminaries

- We start from a source vertex, s.
- Discover vertices in the graph
- Three possible color values of a vertex
  - white, not discovered yet
  - gray, discovered but not explored
  - black, discovered and explored
- Initially, all vertices are white
- The source vertex is discovered first

Predecessor/Parent

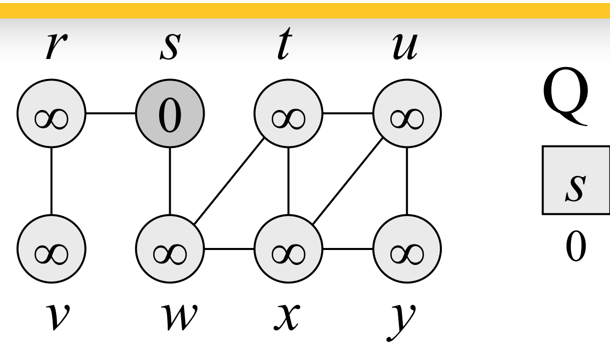
- **While exploring (gray) node u, we check the adjacency list u.Adj**
- **If v on u.Adj is white, we say v is discovered by u. We also say u is the parent or predecessor of v. This is denoted by v.π=u**

Breadth-First-Search

```
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 vertex 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
  }
```

Running Example

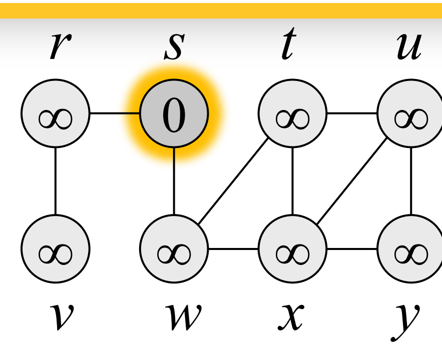
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

```
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 vertex 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
```

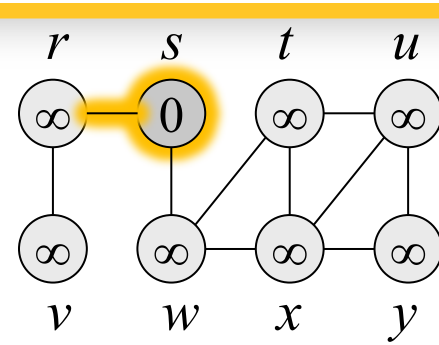


Q

Assume that the adjacency lists are alphabetical

Running Example

```
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 vertex 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
```



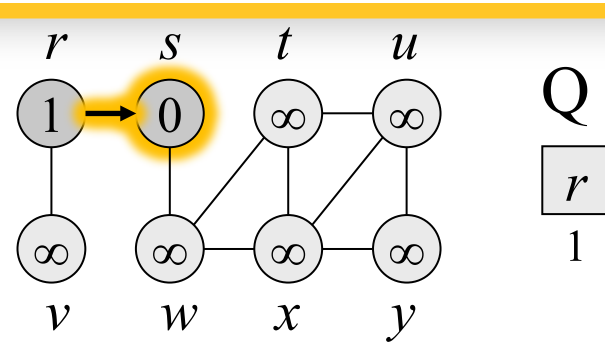
Q

Assume that the adjacency lists are alphabetical

Running Example

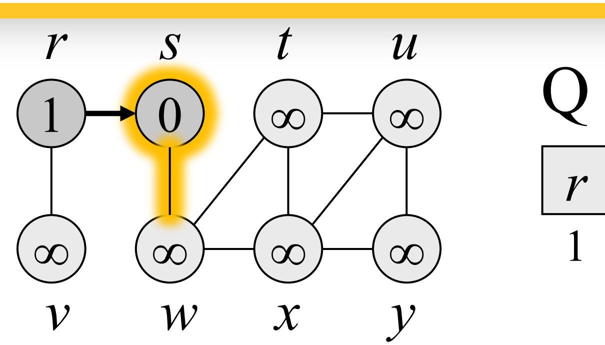
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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

Assume that the adjacency lists are alphabetical



Running Example

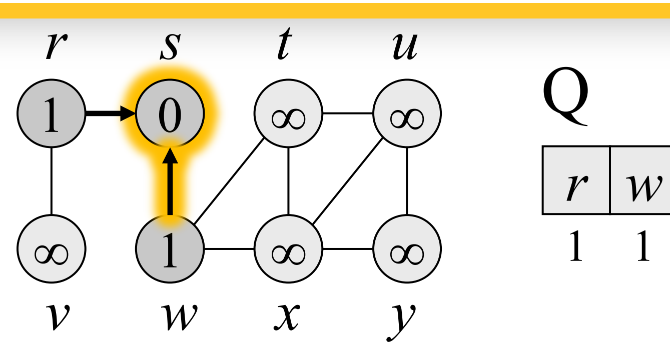
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

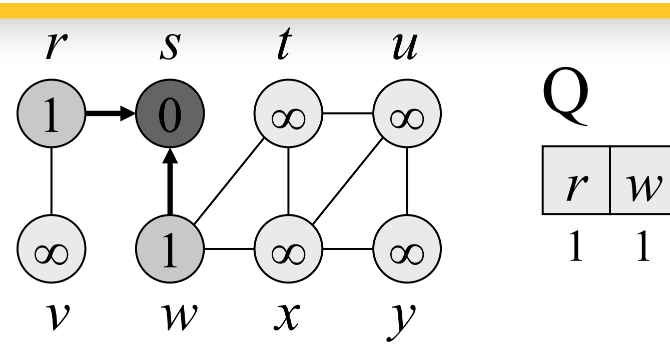
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

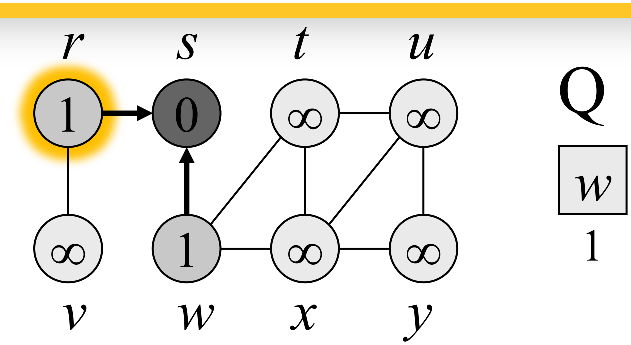
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

```
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 vertex 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
```

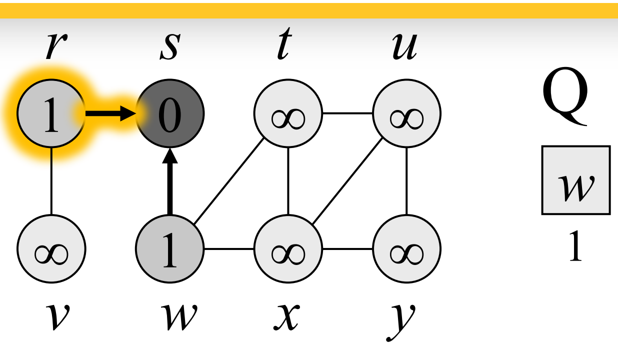


Assume that the adjacency lists are alphabetical

Running Example

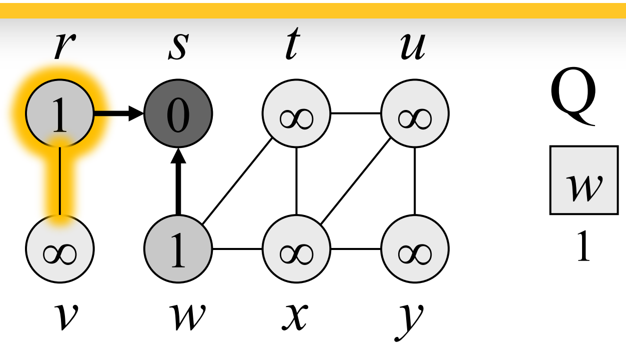
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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

Assume that the adjacency lists are alphabetical



Running Example

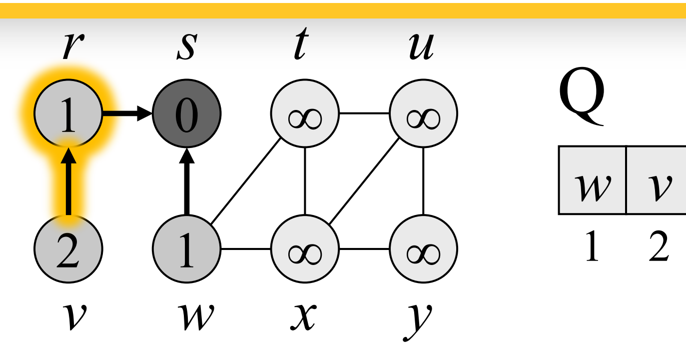
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

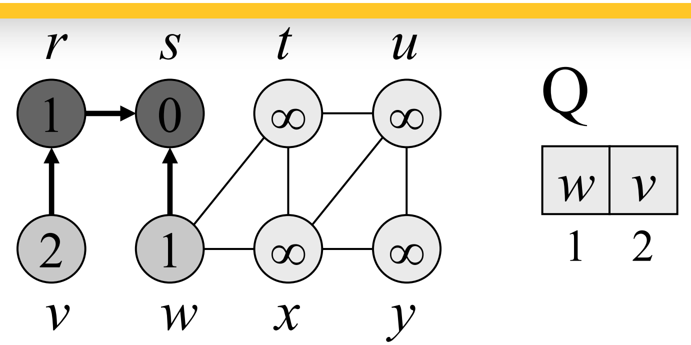
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

```
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 vertex 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
```

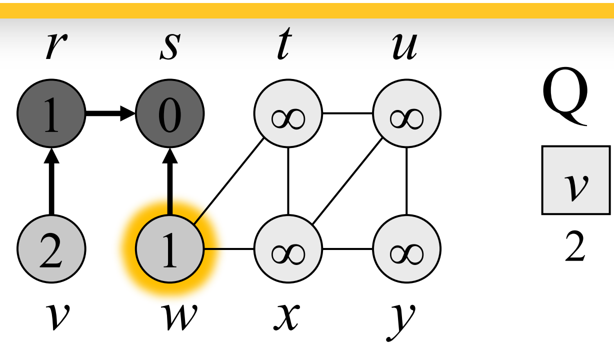


Assume that the adjacency lists are alphabetical

Running Example

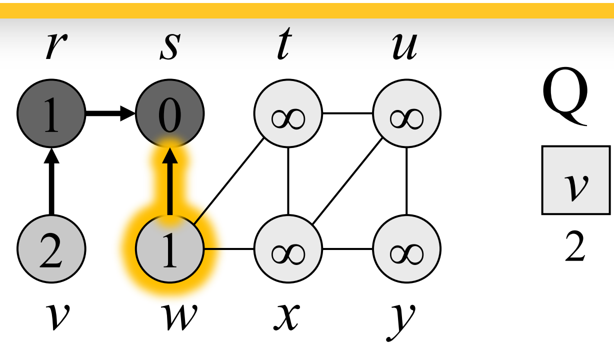
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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

Assume that the adjacency lists are alphabetical



Running Example

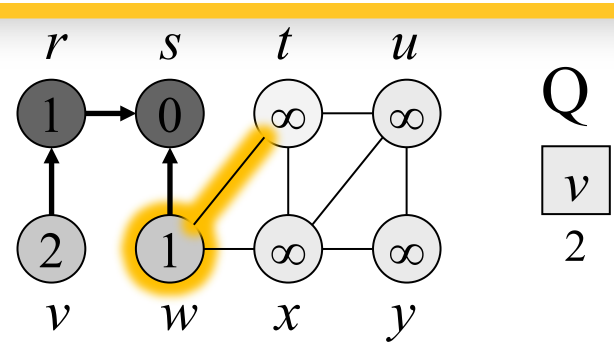
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

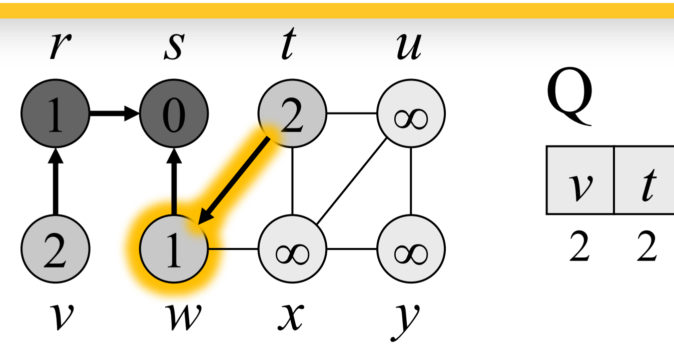
Running Example

```
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 vertex 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
```



Assume that the adjacency lists are alphabetical

Running Example

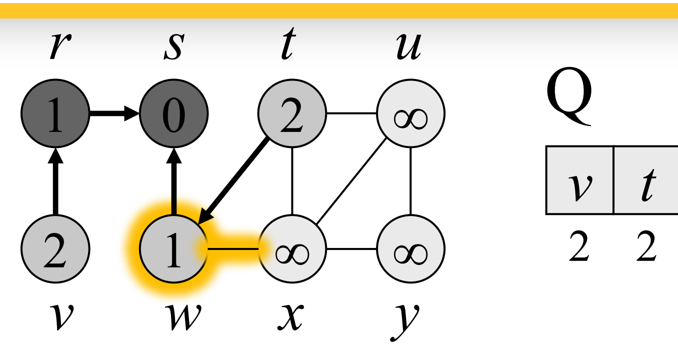



```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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

Assume that the adjacency lists are alphabetical

Running Example

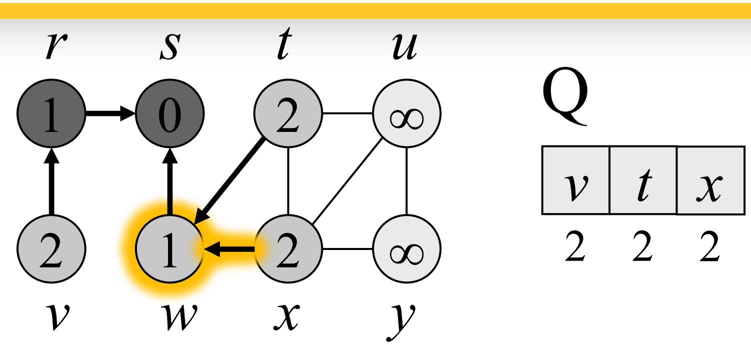
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

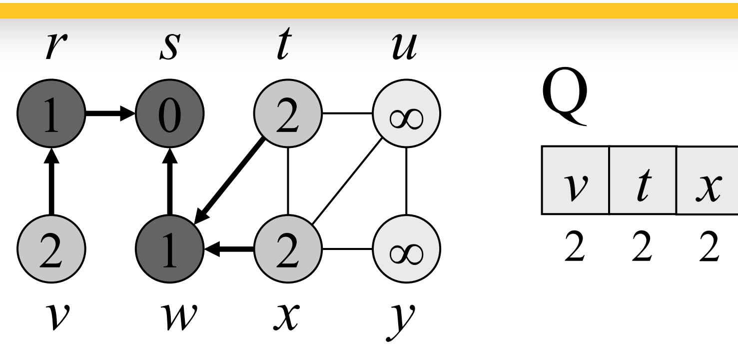
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

```
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 vertex 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
```

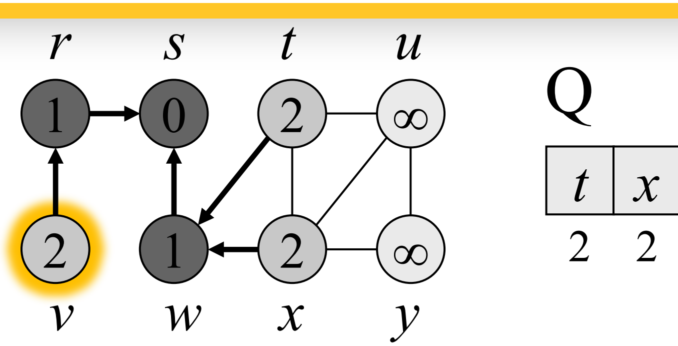


Assume that the adjacency lists are alphabetical

Running Example

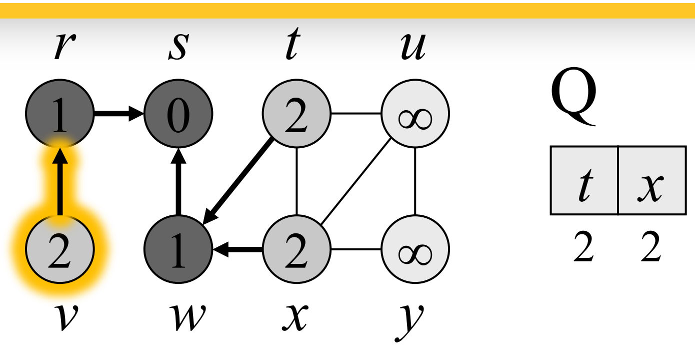
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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

Assume that the adjacency lists are alphabetical



Running Example

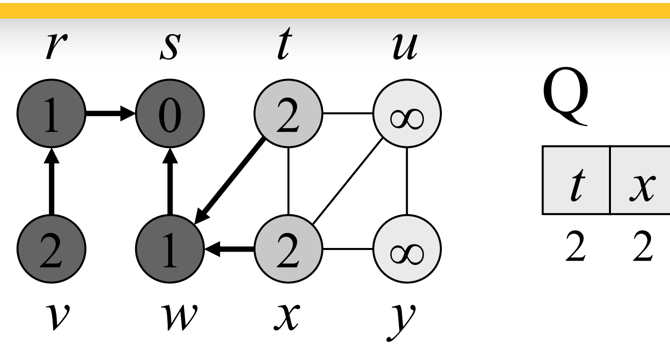
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

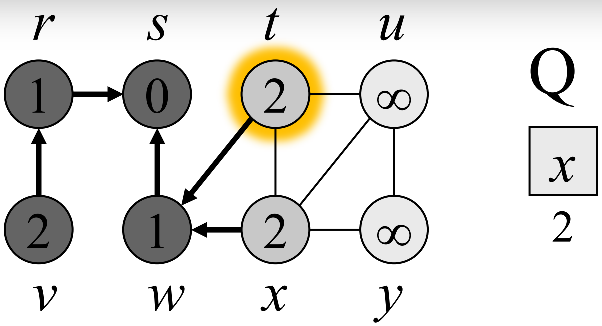
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

```
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 vertex 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
```

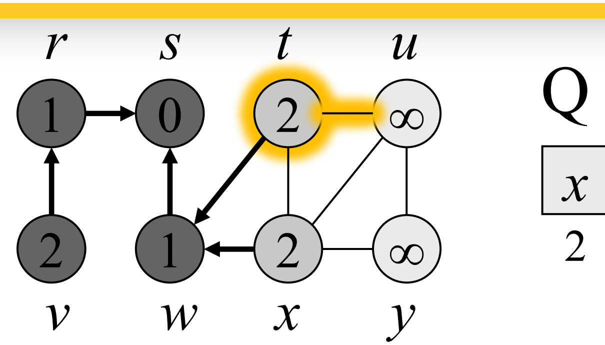


Assume that the adjacency lists are alphabetical

Running Example

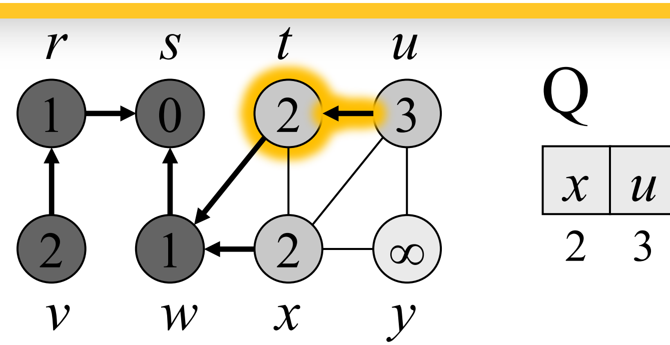
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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

Assume that the adjacency lists are alphabetical



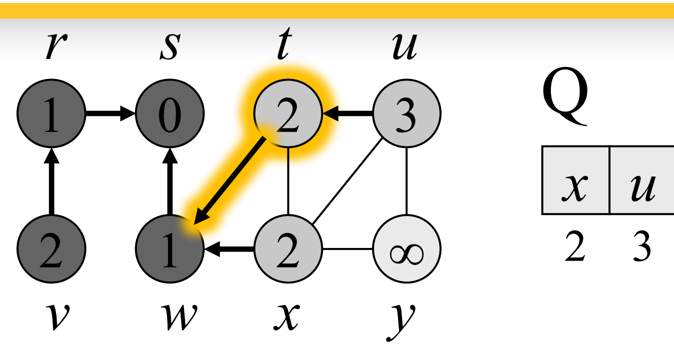
Running Example

```
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 vertex 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
```



Assume that the adjacency lists are alphabetical

Running Example

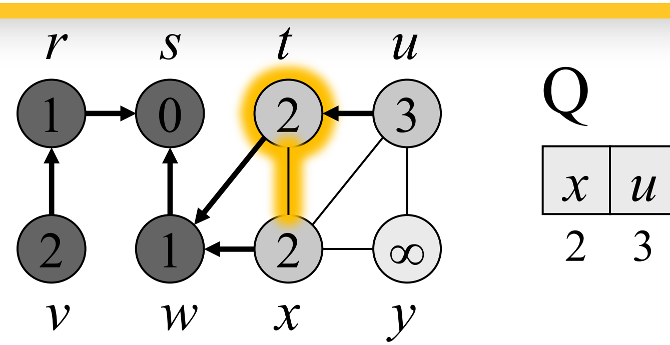



```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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

Assume that the adjacency lists are alphabetical

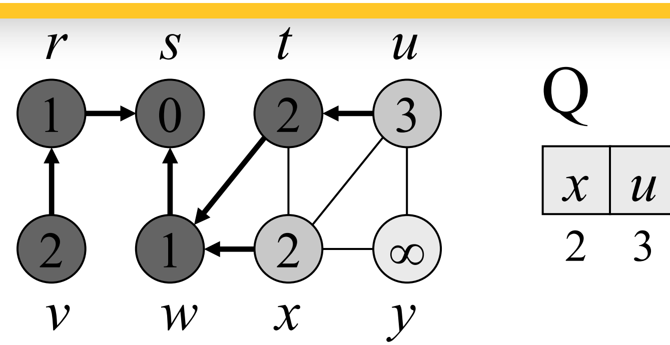
Running Example

```
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 vertex 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
```



Assume that the adjacency lists are alphabetical

Running Example

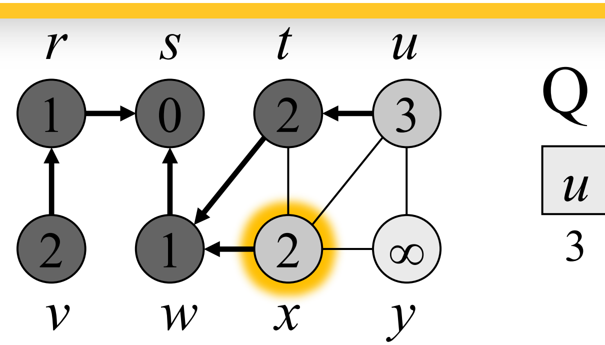



```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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

Assume that the adjacency lists are alphabetical

Running Example

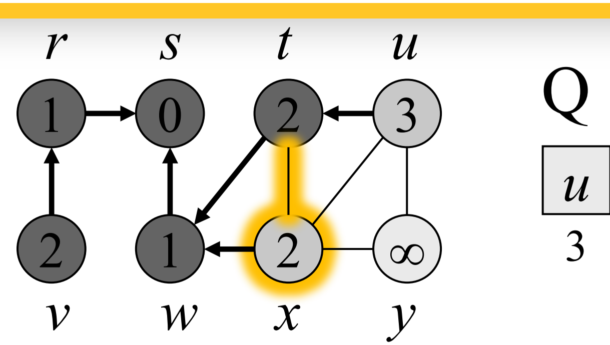
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

```
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 vertex 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
```

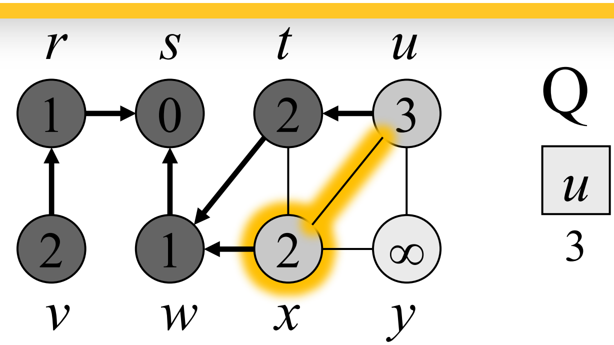


Assume that the adjacency lists are alphabetical

Running Example

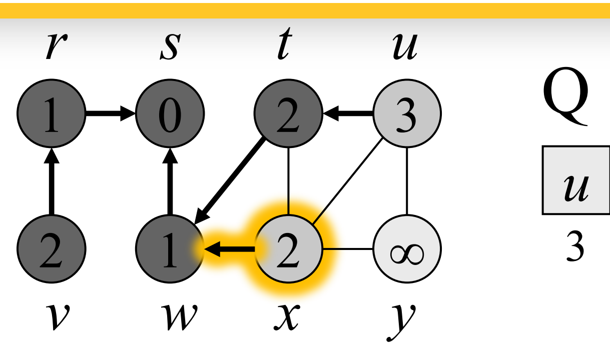
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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

Assume that the adjacency lists are alphabetical



Running Example

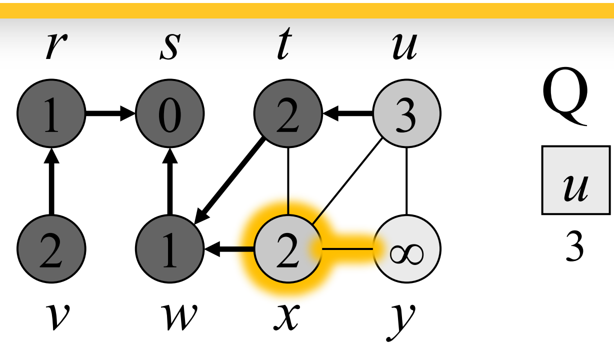
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

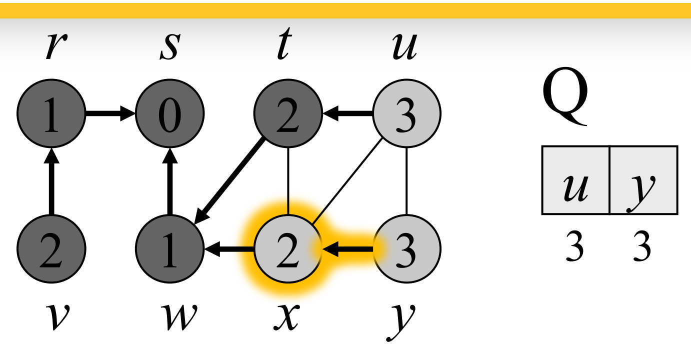
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

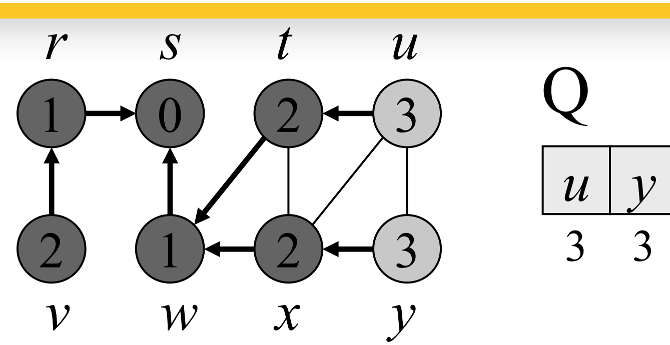
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

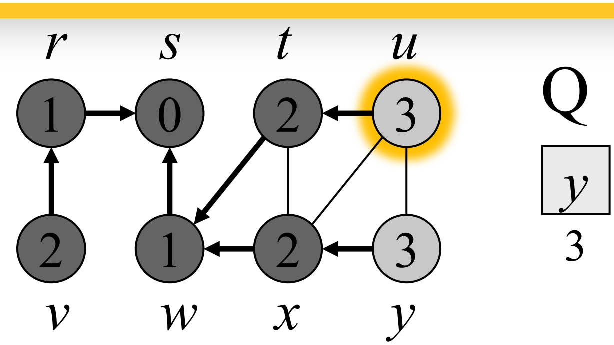
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

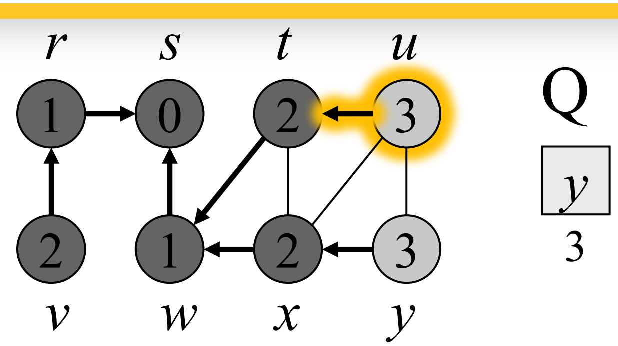
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

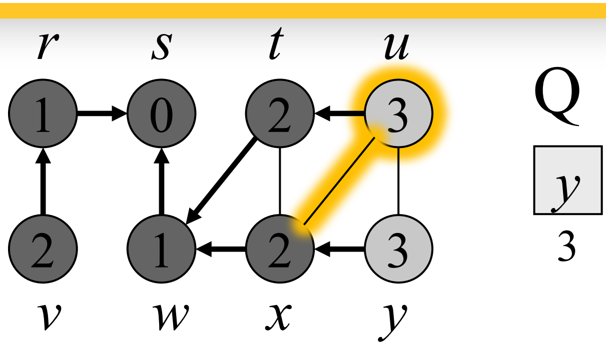
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

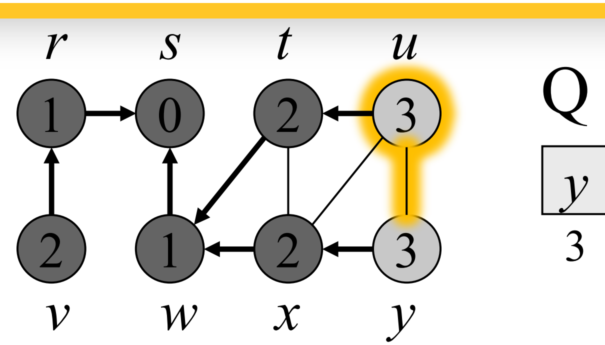
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

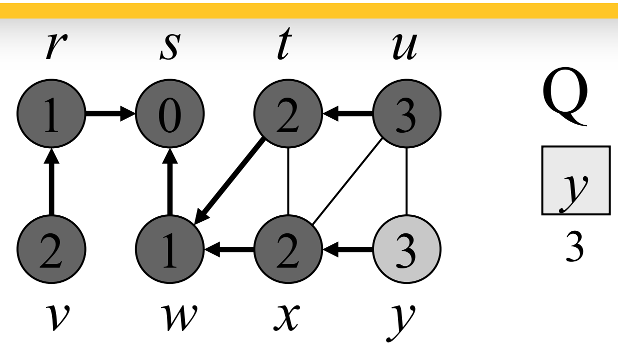
```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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



Assume that the adjacency lists are alphabetical

Running Example

```
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 vertex 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
```

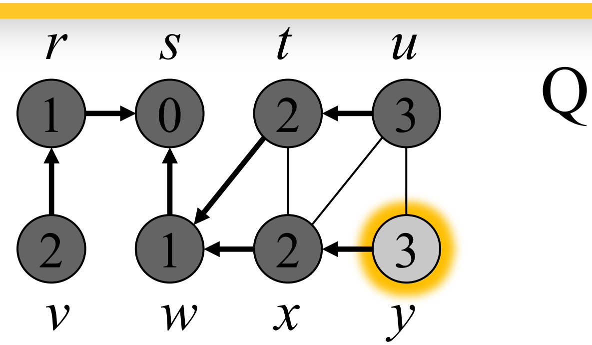


Assume that the adjacency lists are alphabetical

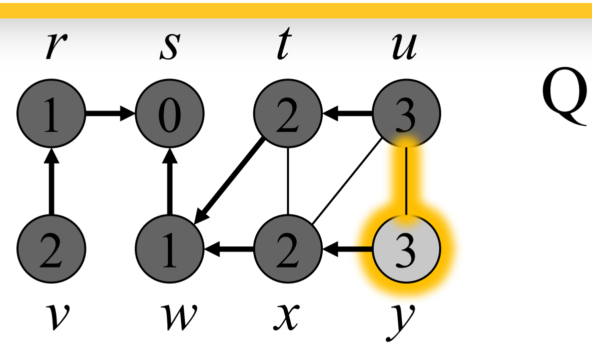
Running Example

```
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 vertex 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
```

Assume that the adjacency lists are alphabetical



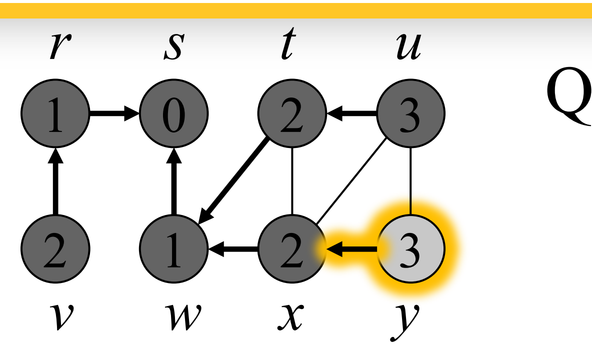
Running Example




```
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 vertex 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
```

Assume that the adjacency lists are alphabetical

Running Example



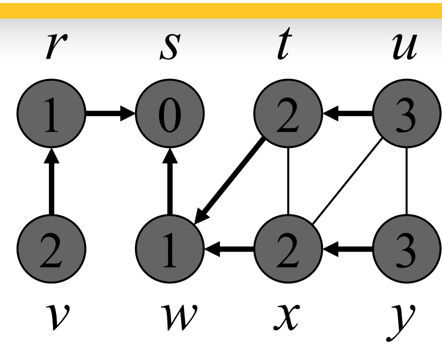

```
BFS(G, s)
  for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
  s.color = GRAY
  s.d = 0
  s. π = NIL
  Q = ∅
  Enqueue(Q, s)
  while (Q≠ ∅)
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u]
      if v.color == WHITE
        v.color = GRAY
        v.d = u.d + 1
      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

Assume that the adjacency lists are alphabetical

Running Example

```
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 vertex 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
```

Assume that the adjacency lists are alphabetical



Q

BFS on Undirected and Directed Graphs

The BFS algorithm can be applied to both undirected graphs and directed graphs.

Ira A. Fulton Schools of
Engineering
Arizona State University

Graphs, Part 4

Ira A. Fulton Schools of Engineering
Arizona State University

Running Time of BFS

```
BFS(G, s)
1   for each vertex u ∈ G.V − {s}
2     u.color = WHITE
3     u.d = ∞
4     u.π = NIL
5   s.color = GRAY
6   s.d = 0
7   s. π = NIL
8   Q = ∅
9   Enqueue(Q, s)
10  while (Q≠ ∅)
11    u = DEQUEUE(Q)
12    for each vertex v ∈ G.Adj[u]
13      if v.color == WHITE
14        v.color = GRAY
15        v.d = u.d + 1
16        v.π = u
17        Enqueue(Q, v)
18    u.color = BLACK
```

Theorem: The running time of BFS is O(n+m)

Proof. Lines 1-9 takes O(n) time.

The while-loop takes O(n+m) time, since each vertex will be added to Q at most once and deleted at most once. Each edge is visited at most twice.

Shortest Paths

Let s be the source vertex, and v any vertex. Define the shortest-path distance $\delta(s, v)$ from s to v as the minimum number of edges in any path from s to v.

If there is no path from s to v, $\delta(s, v) = \infty$.

Lemma 20.1 For any edge $(u, v) \in E$, $\delta(s, v) \leq \delta(s, u) + 1$

Proof. Since $(u, v) \in E$, v is reachable from s if and only if u is reachable from s. The shortest path from s to v cannot be longer than the shortest path from s to u followed by the edge (u, v). This proves the lemma.

Shortest Paths

Lemma 20.2. 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, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies $v.d \geq \delta(s, v)$at all times.

Proof. When $v.d$ is assigned a finite value, it is the length of some path from s to v. Therefore $v.d \geq \delta(s, v)$. The vertex is also colored GRAY, hence $v.d$ will never be changed again.

Shortest Paths

Lemma 20.3. Suppose that during the execution of BFS on a graph G=(V, E), the queue Q contains the vertices $< v_1, v_2, \ldots, v_r >$, where v_1 is the head and v_r is the tail. Then $v_r \leq v_1.d + 1$ and $v_i.d \leq v_{i+1}.d$, $i = 1, 2, \ldots, r-1$

Proof. This can be proved by induction on the number of queue operations.

Theorem 20.5. Suppose we run BFS on graph G=(V, E). Upon termination, we have $v.d = \delta(s, v)$ for all $v \in V$.

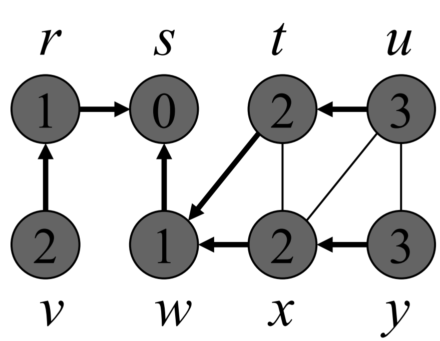
Proof. This can be proved by induction.

Breadth-first Tree

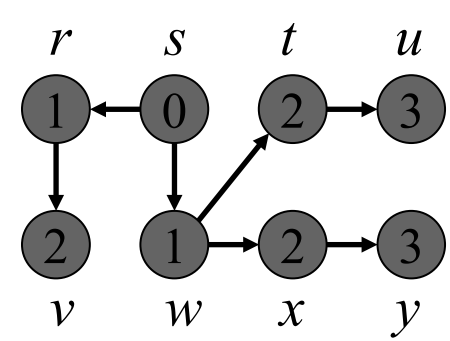
Define the predecessor sub-graph of G as $G_\pi = (V_\pi, E_\pi)$, where

$$V_\pi = \{v \in V : v.\pi \neq NIL\} \cup \{s\}$$
$$E_\pi = \{(v.\pi, v) : v \in V_\pi - \{s\}\}$$

G_π defines the Breadth-first tree.



Breadth-first tree



Layers of Vertices

- While performing BFS, we also compute layers of the vertices
- $L_i = \{u \in V \mid u.d = i\}$.
- $L_0 = \{s\}$
- $L_1 = \{r, w\}$
- $L_2 = \{t, v, x\}$
- $L_3 = \{u, y\}$

ASU®

Ira A. Fulton Schools of
Engineering

Arizona State University