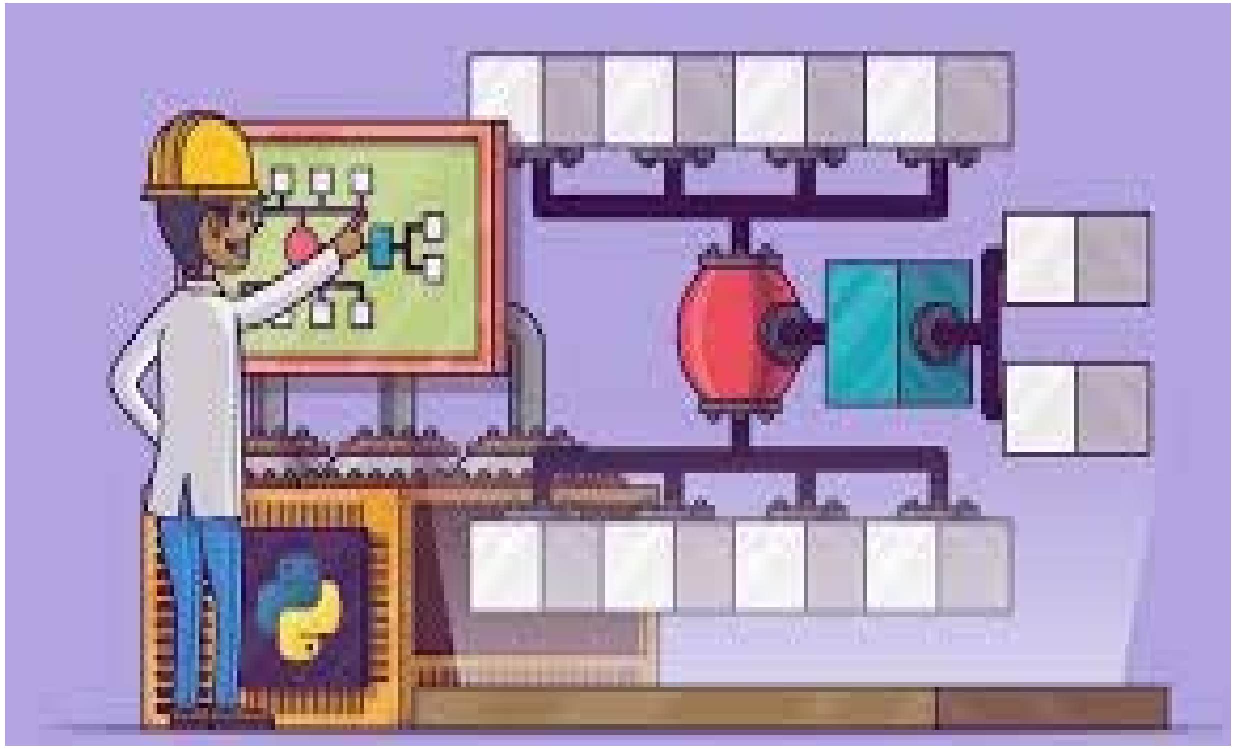
ساختمان داده ها



مدرس:
سمانه حسینی سمنانی

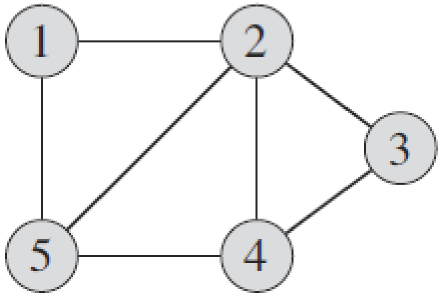
دانشگاه صنعتی اصفهان- دانشکده برق و کامپیوتر

Elementary Graph Algorithms

- Graph representation
- graph-searching algorithm
  - breadth-first search
  - depth-first search
- minimum-spanning-tree
  - Kruskal
  - Prim
  - Sollin

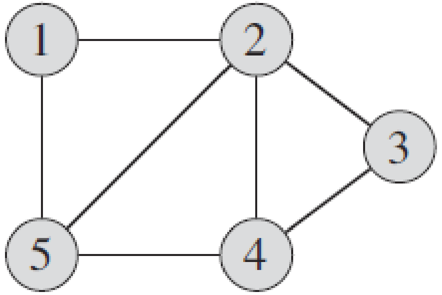
Representations of graphs

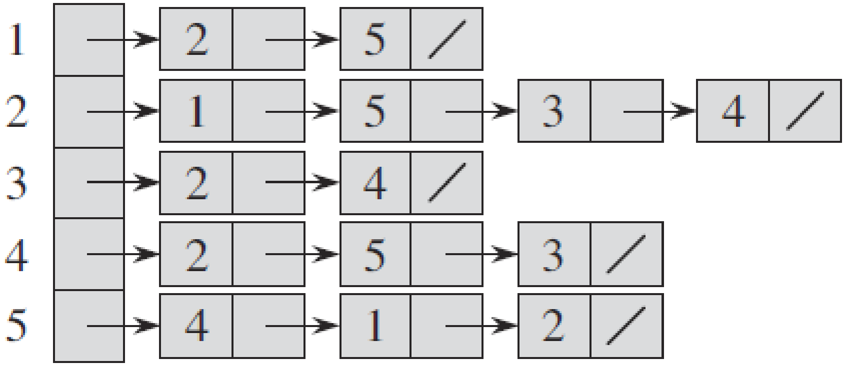
- collection of adjacency lists good for **sparse** graphs
- adjacency matrix good for **dense** graphs



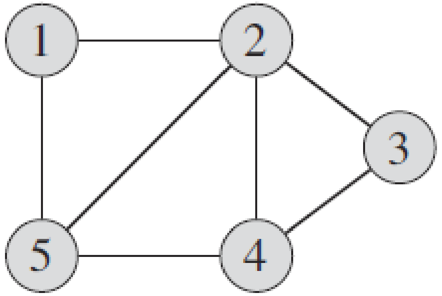
An undirected graph G with 5 vertices and 7 edges.

Collection of adjacency lists





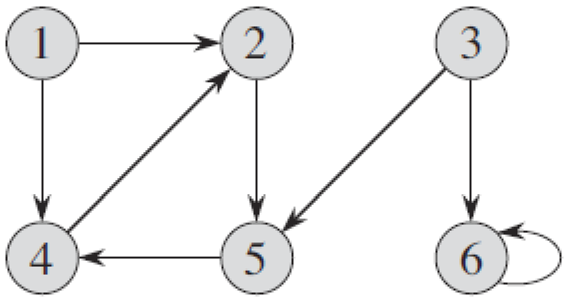
Adjacency matrix



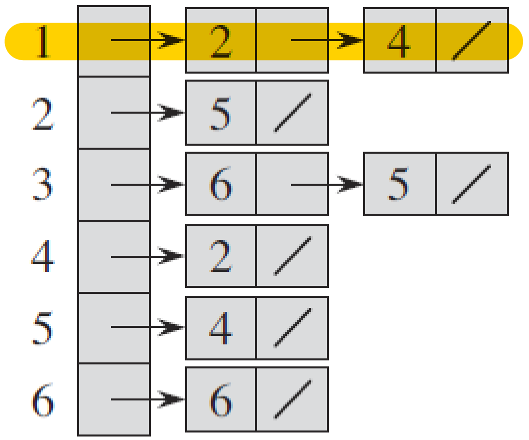
$$
\begin{array}{c|ccccc|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
$$

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

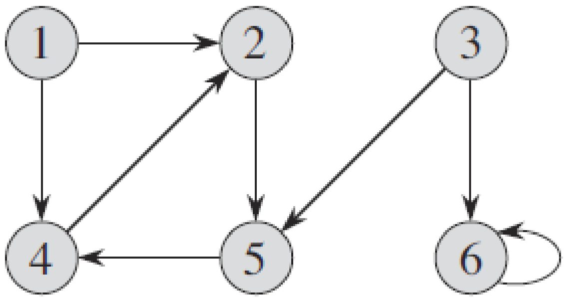
Collection of adjacency lists



A directed graph G with 6 vertices and 8 edges



Collection of adjacency lists



	1	2	3	4	5	6
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

Collection of adjacency lists

- If G is a directed graph, the sum of the lengths of all the adjacency lists is $|E|$
- If G is an undirected graph, the sum of the lengths of all the adjacency lists is $2|E|$
- For both directed and undirected graphs, the adjacency-list representation has the desirable property that the amount of memory it requires is $\Theta(V + E)$

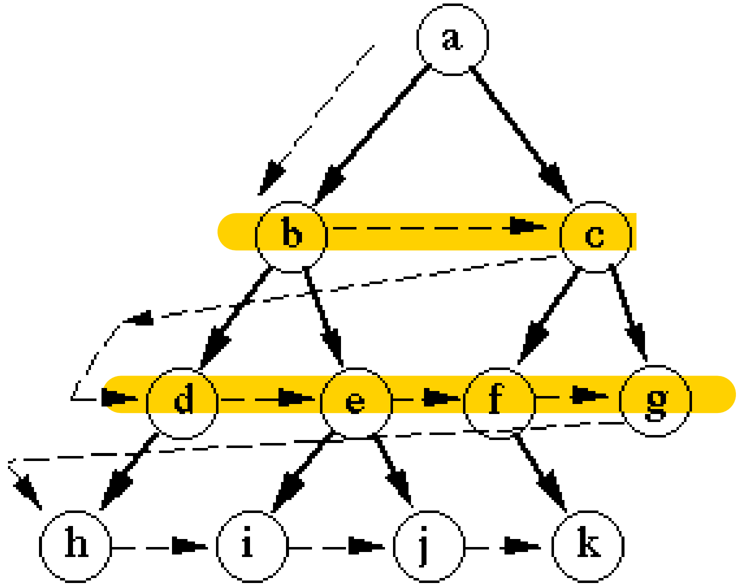
Weighted graphs

- adjacency matrix can represent a weighted graph
- We simply store the weight $w(u, v)$ of the edge $(u, v) \in E$ with vertex v in u's adjacency list.
- If an edge does not exist, we can store a NIL value as its corresponding matrix entry, though for many problems it is convenient to use a value such as ∞.
- The adjacency-list representation is quite robust in that we can modify it to support many other graph variants.
- A potential disadvantage of the adjacency-list representation is that it provides no quicker way to determine whether a given edge (u, v) is present in the graph than to search for v in the adjacency list $Adj[u]$.

Breadth-first search

- one of the simplest algorithms
- archetype for many important graph algorithms e.g Prim, Dijkstra
- Given a graph $G = (V, E)$ and a distinguished source vertex s
- Breadth-first search systematically explores the edges of G to “discover” every vertex that is reachable from s.
- Breadth-first search is so named because it discovers all vertices at distance k from s before discovering any vertices at distance $k + 1$.

Breadth-first search



Breadth-first search

Breadth-first search

- Breadth-first search colors each vertex white, gray, or black.
- All vertices start out white and may later become gray and then black.
- A vertex is discovered the first time it is encountered during the search, at which time it becomes nonwhite.
- Gray and black vertices, therefore, have been discovered, but breadth-first search distinguishes between them to ensure that the search proceeds in a breadth-first manner.

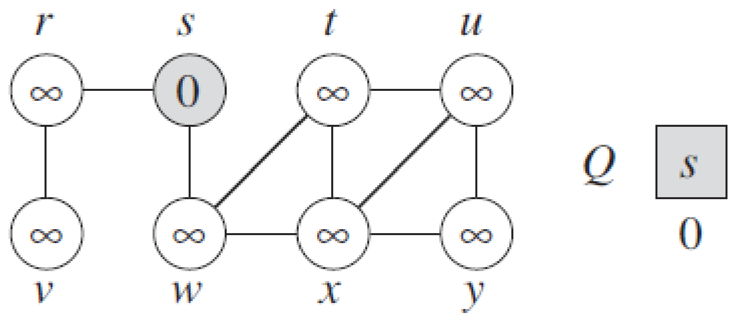
Breadth-first search

- $u.color$
- $u.\pi$ predecessor of u.
- $u.d$ holds the distance from the source s to vertex u:
  - number of edges in any path from vertex s to vertex v.
- use a first-in, first-out queue Q to manage the set of gray vertices.

Breadth-first search

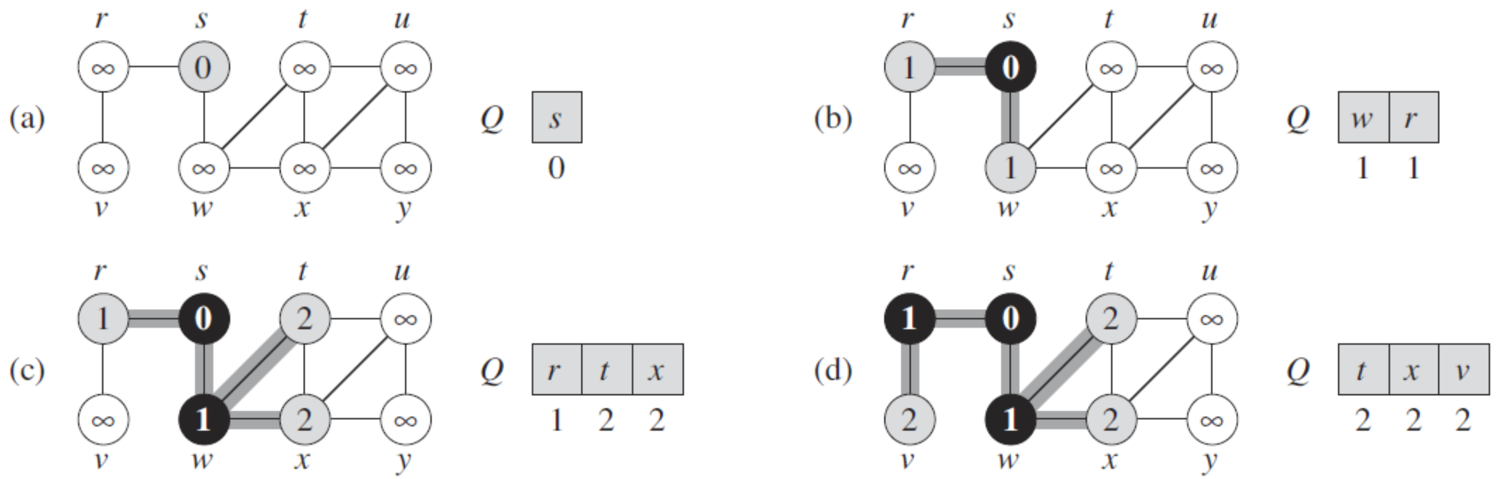
```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GR
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q
while Q ≠ ∅
    u = DE
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q,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

(a)

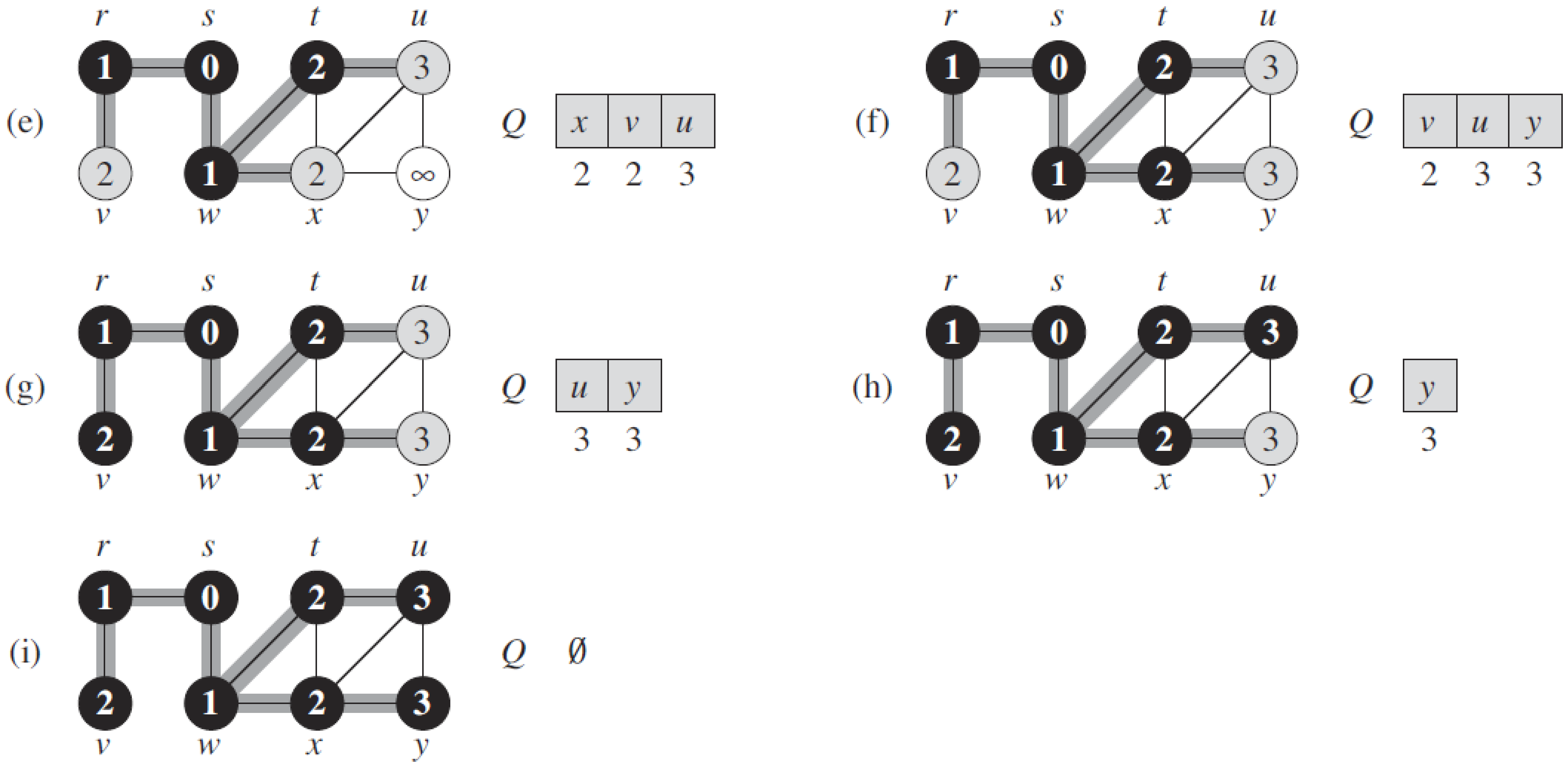


Breadth-first search

```
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 = Ø
ENQUEUE(Q, s)
while Q ≠ Ø
    u = DEQUEUE(Q)
    for each 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
```



Breadth-first search



Breadth-first search

- The results of breadth-first search may depend upon the order in which the neighbors of a given vertex are visited in line 12

- the breadth-first tree may vary, but the distances d computed by the algorithm will not

```
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 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
```

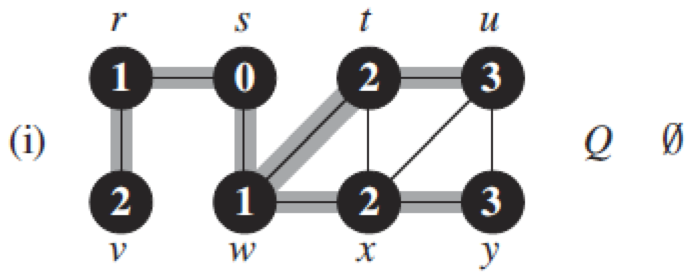
BFS Analysis

- هر راس ملاقات شده فقط یکبار داخل صف قرار می گیرد.
- ماتریس مجاورتی: $O(n^2)$
- لیست مجاورتی:

- The sum of the lengths of all the adjacency lists is $O(E)$
- The overhead for initialization is $O(V)$
- $O(V+E)$.

Shortest paths

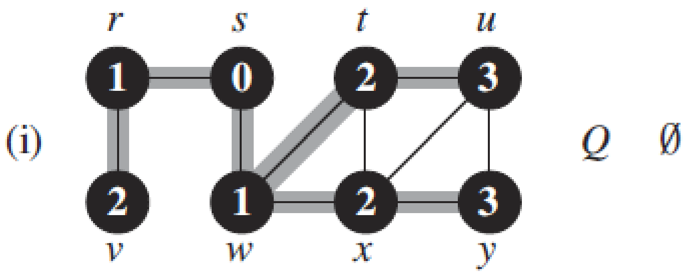
- shortest-path distance $\delta(s, v)$ from s to v as the minimum number of edges in any path from vertex s to vertex v.
- if there is no path from s to v then $\delta(s, v) = \infty$.
- We can show that breadth-first search correctly computes shortest path distances,



Breadth-first trees

- The procedure BFS builds a breadth-first tree as it searches the graph

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



linear in the number of vertices in the path printed