

# Breadth First Search

CSE 2011  
Fall 2009

11/29/2009 6:02 PM

1

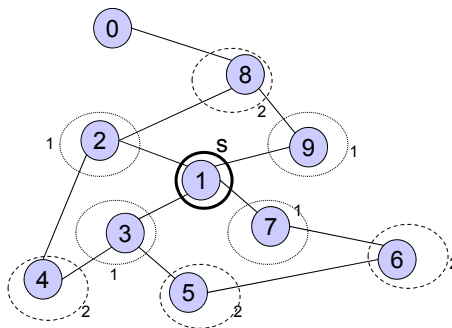
## Graph Traversal

- Application example
  - Given a graph representation and a vertex  $s$  in the graph, find all paths from  $s$  to the other vertices.
- Two common graph traversal algorithms:
  - Breadth-First Search (BFS)
    - Idea is similar to level-order traversal for trees.
    - Implementation uses a queue.
    - Gives shortest path from a vertex to another.
  - Depth-First Search (DFS)
    - Idea is similar to preorder traversal for trees (visit a node then visit its children recursively).
    - Implementation uses a stack (implicitly via recursion).

2

## BFS and Shortest Path Problem

- Given any source vertex  $s$ , BFS visits the other vertices at increasing distances away from  $s$ . In doing so, BFS discovers shortest paths from  $s$  to the other vertices.
- What do we mean by “distance”? The number of edges on a path from  $s$  (unweighted graph).



Example

Consider  $s$ =vertex 1

Nodes at distance 1?  
2, 3, 7, 9

Nodes at distance 2?  
8, 6, 5, 4

Nodes at distance 3?  
0

3

## How Does BFS Work?

- Similarly to level-order traversal for trees.
- Code: similar to code of topological sort.
  - $flag[v] = \text{false}$ : we have not visited  $v$
  - $flag[v] = \text{true}$ : we already visited  $v$
- The BFS code we will discuss works for both directed and undirected graphs.

4

## Skeleton of BFS Algorithm

**Algorithm**  $BFS(s)$

**Input:**  $s$  is the source vertex

**Output:** Mark all vertices that can be visited from  $s$ .

$Q$  = empty queue;

$enqueue(Q, s)$ ;

**while**  $Q$  is not empty

**do**  $v := dequeue(Q)$ ; output  $v$ ;

**for** each  $w$  adjacent to  $v$

$enqueue(Q, w)$

5

## BFS Algorithm

**Algorithm**  $BFS(s)$

**Input:**  $s$  is the source vertex

**Output:** Mark all vertices that can be visited from  $s$ .

1. **for** each vertex  $v$

2.     **do**  $flag[v] := \text{false}$ ;

**flag[ ]: visited or not**

3.  $Q$  = empty queue;

4.  $flag[s] := \text{true}$ ;

5.  $enqueue(Q, s)$ ;

6. **while**  $Q$  is not empty

7.     **do**  $v := dequeue(Q)$ ; output  $v$ ;

8.         **for** each  $w$  adjacent to  $v$

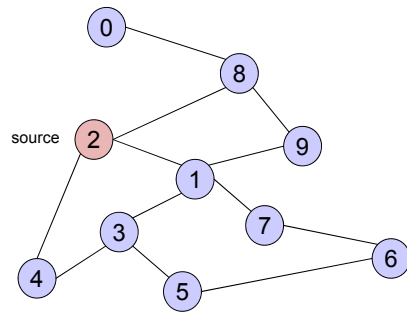
9.             **do if**  $flag[w] = \text{false}$

10.                 **then**  $flag[w] := \text{true}$ ;

11.                      $enqueue(Q, w)$

6

# BFS Example



$Q = \{ \}$

Initialize  $Q$  to be empty

Adjacency List

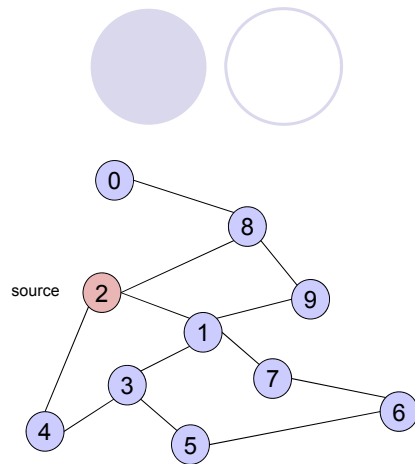
0	8
1	3 7 9 2
2	8 1 4
3	4 5 1
4	2 3
5	3 6
6	7 5
7	1 6
8	2 0 9
9	1 8

Visited Table (T/F)

0	F
1	F
2	F
3	F
4	F
5	F
6	F
7	F
8	F
9	F

Initialize "visited" table (all False)

7



$Q = \{ 2 \}$

Place source 2 on the queue

Adjacency List

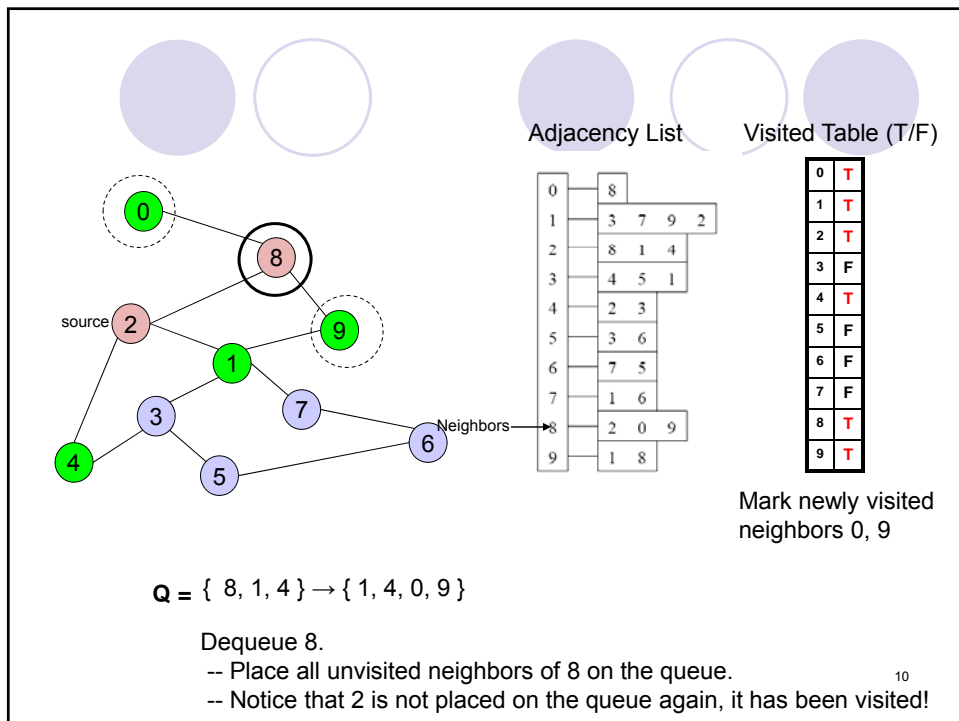
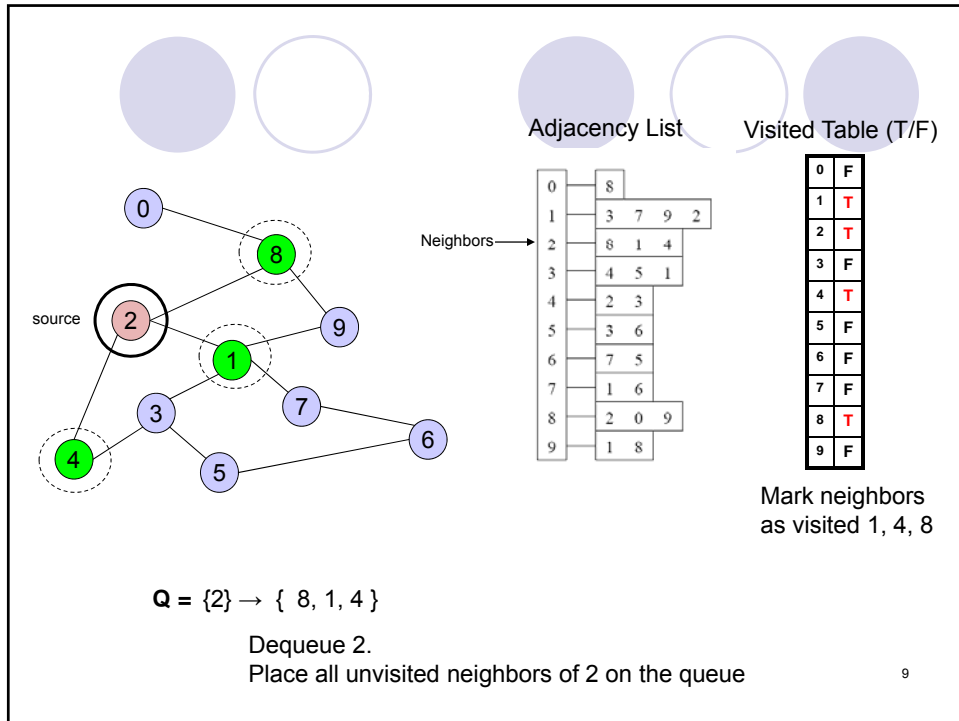
0	8
1	3 7 9 2
2	8 1 4
3	4 5 1
4	2 3
5	3 6
6	7 5
7	1 6
8	2 0 9
9	1 8

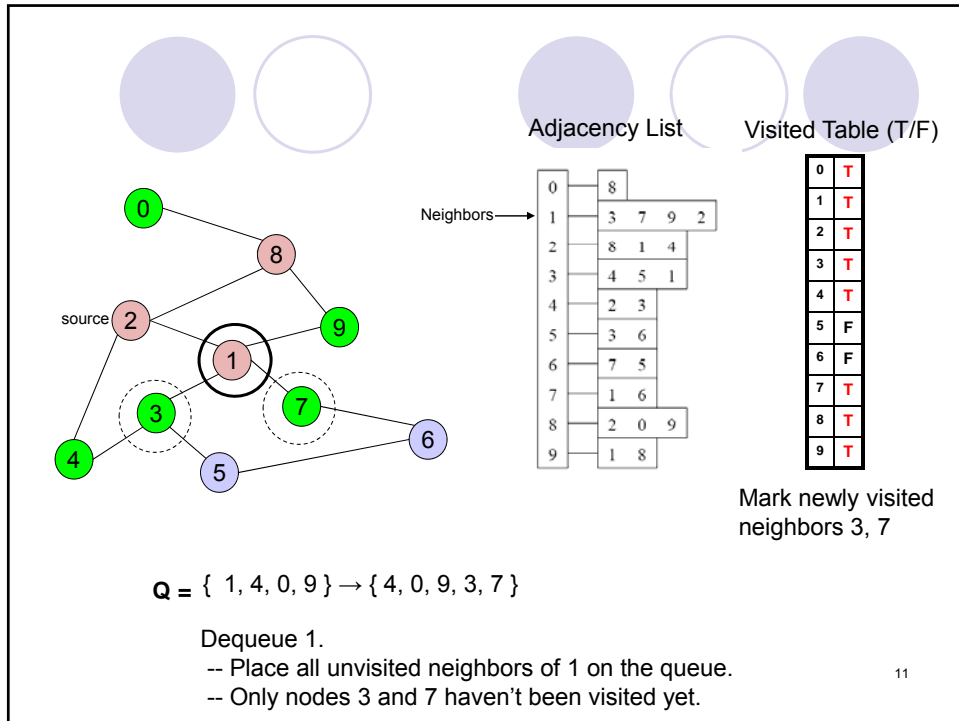
Visited Table (T/F)

0	F
1	F
2	T
3	F
4	F
5	F
6	F
7	F
8	F
9	F

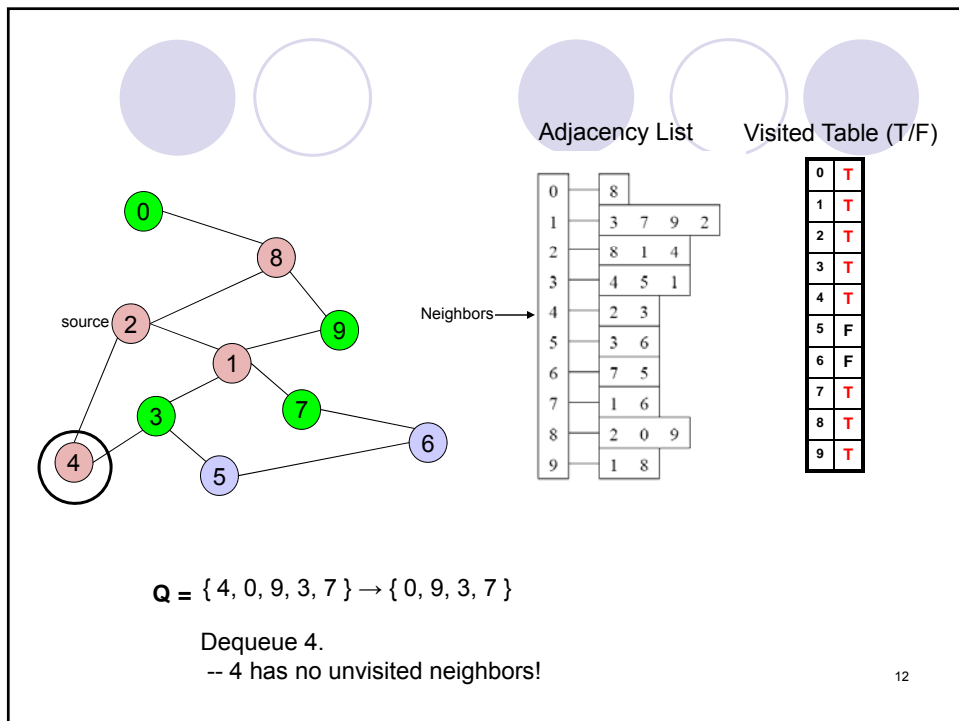
Flag that 2 has been visited

8

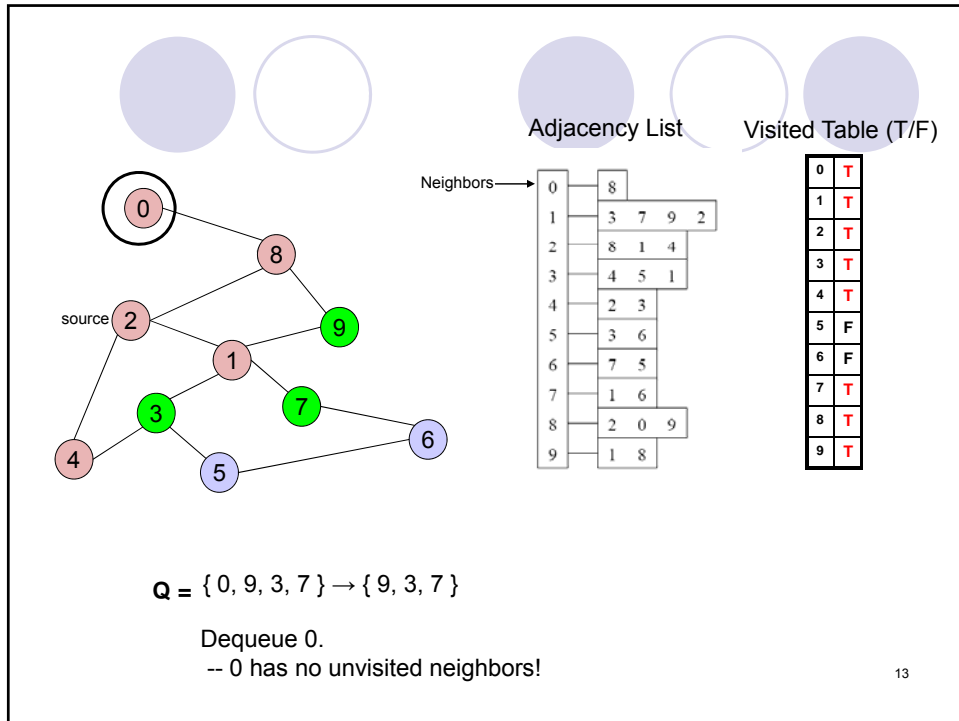




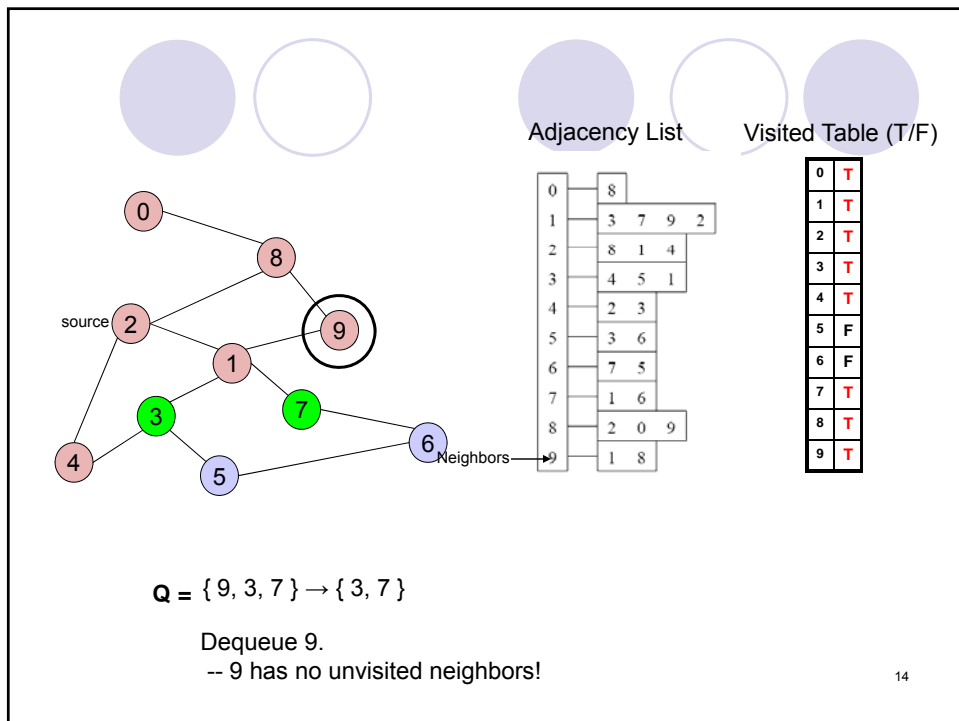
11



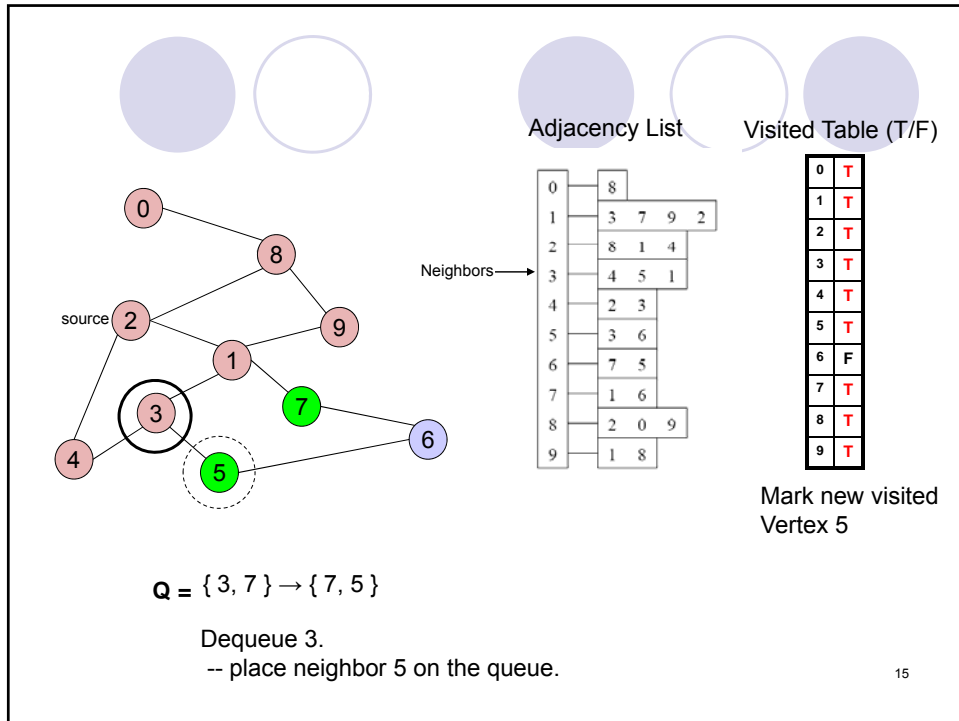
12



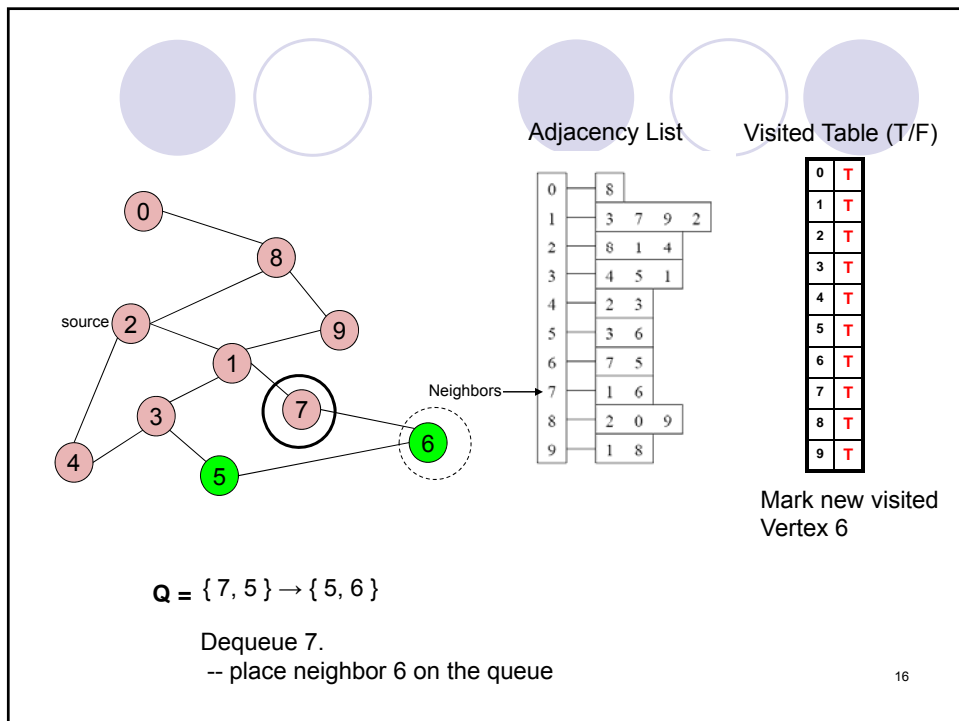
13



14

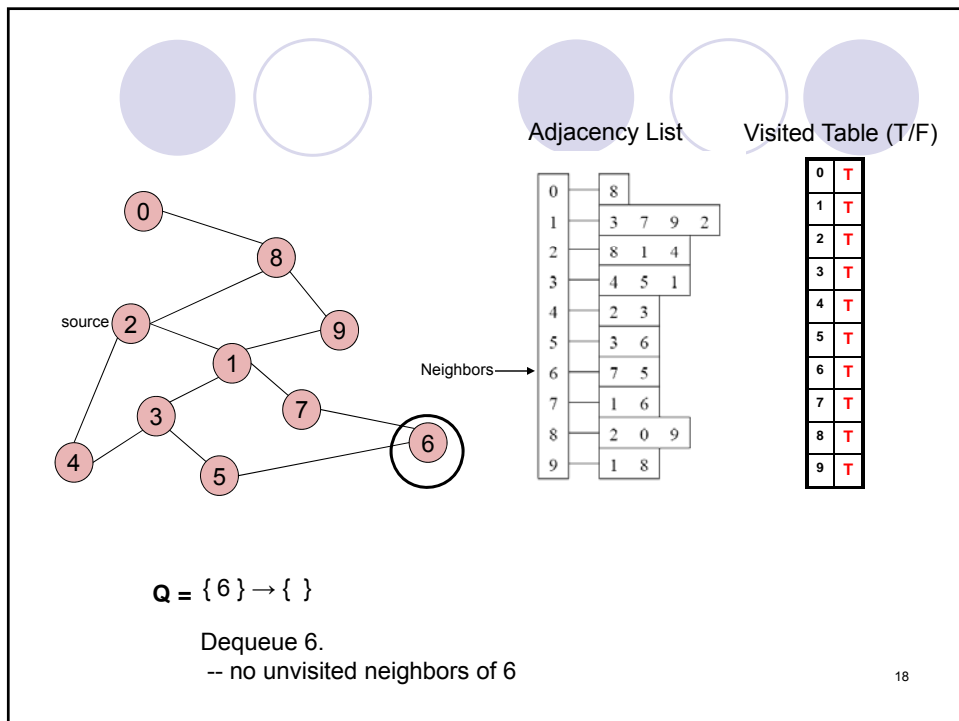
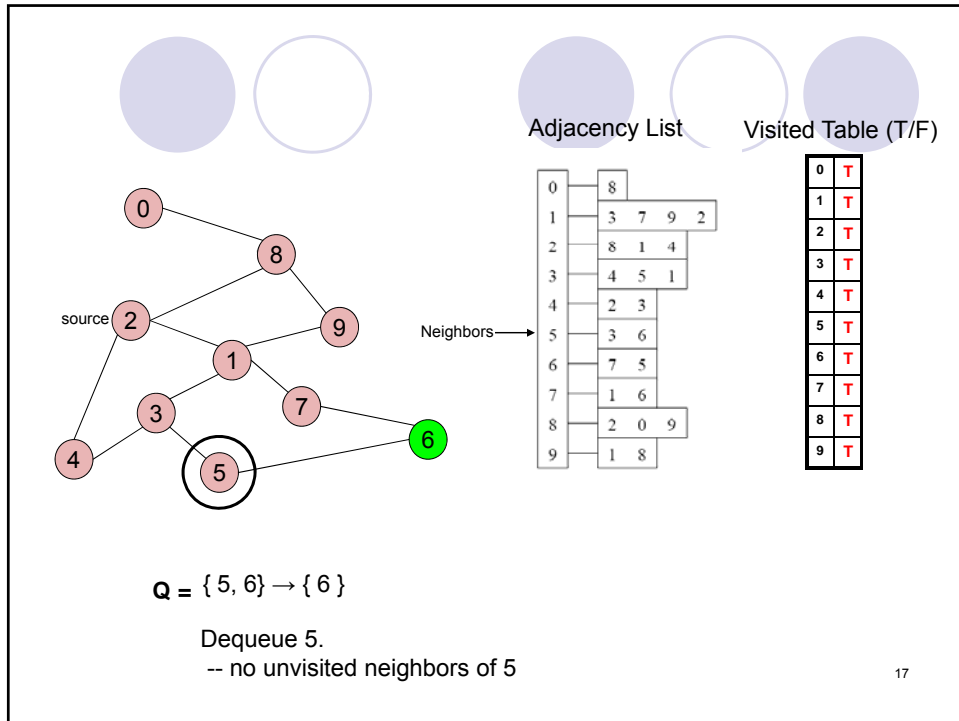


15



16





source 2

Adjacency List

0	8
1	3 7 9 2
2	8 1 4
3	4 5 1
4	2 3
5	3 6
6	7 5
7	1 6
8	2 0 9
9	1 8

Visited Table (T/F)

0	T
1	T
2	T
3	T
4	T
5	T
6	T
7	T
8	T
9	T

**Q = { } STOP!!! Q is empty!!!**

What did we discover?

Look at "visited" tables.

There exists a path from source vertex 2 to all vertices in the graph

## Running Time of BFS

- Assume adjacency list
  - V = number of vertices; E = number of edges

**Algorithm *BFS(s)***

**Input:** *s* is the source vertex

**Output:** Mark all vertices that can be visited from *s*.

1. **for** each vertex *v*
2.     **do** *flag[v]* := false;
3. *Q* = empty queue;
4. *flag[s]* := true;
5. *enqueue(Q, s)*;
6. **while** *Q* is not empty
7.     **do** *v* := *dequeue(Q)*;
8.     **for** each *w* adjacent to *v*
9.         **do if** *flag[w]* = false
10.             **then** *flag[w]* := true;
11.             *enqueue(Q, w)*

Each vertex will enter *Q* at most once. *dequeue* is  $O(1)$ .

The for loop takes time proportional to  $\deg(v)$ .

20

## Running Time of BFS (2)

- Recall: Given a graph with E edges

$$\sum_{\text{vertex } v} \deg(v) = 2E$$

- The total running time of the while loop is:

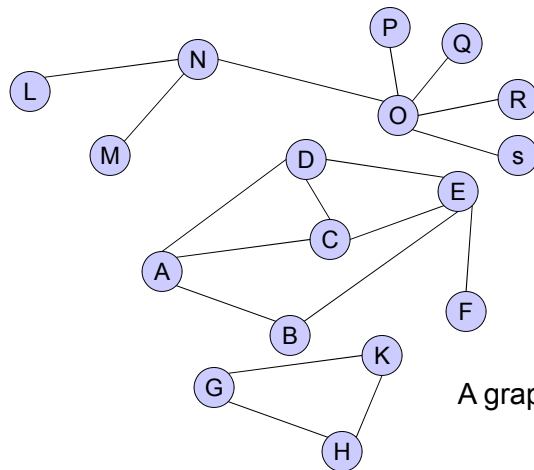
$$O\left(\sum_{\text{vertex } v} (1 + \deg(v))\right) = O(V+E)$$

- This is the sum over all the iterations of the while loop!
- Homework: What is the running time of BFS if we use an adjacency matrix?

21

## BFS and Unconnected Graphs

A graph may not be connected (strongly connected)  $\Rightarrow$  enhance the above BFS code to accommodate this case.



A graph with 3 components

22

## Recall the BFS Algorithm ...

**Algorithm**  $BFS(s)$

**Input:**  $s$  is the source vertex

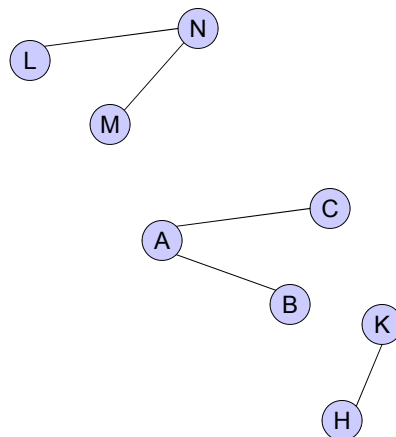
**Output:** Mark all vertices that can be visited from  $s$ .

1. **for** each vertex  $v$
2.     **do**  $flag[v] := false$ ;
3.  $Q =$  empty queue;
4.  $flag[s] := true$ ;
5.  $enqueue(Q, s)$ ;
6. **while**  $Q$  is not empty
7.     **do**  $v := dequeue(Q)$ ;   output (  $v$  );
8.     **for** each  $w$  adjacent to  $v$
9.         **do if**  $flag[w] = false$
10.             **then**  $flag[w] := true$ ;
11.              $enqueue(Q, w)$

23

## Enhanced BFS Algorithm

A graph with 3 components



- We can re-use the previous  $BFS(s)$  method to compute the connected components of a graph  $G$ .

```

BFSearch(  $G$  ) {
   $i = 1$ ;   // component number
  for every vertex  $v$ 
     $flag[v] = false$ ;
  for every vertex  $v$ 
    if (  $flag[v] == false$  ) {
      print ( "Component " +  $i++$  );
      BFS(  $v$  );
    }
}
  
```

24

## Applications of BFS

What can we do with the BFS code we just discussed?

- Is there a path from source  $s$  to a vertex  $v$ ?
  - Check  $flag[v]$ .
- Is an undirected graph connected?
  - Scan array  $flag[]$ .
  - If there exists  $flag[u] = \text{false}$  then ...
- Is a directed graph strongly connected?
  - Scan array  $flag[]$ .
  - If there exists  $flag[u] = \text{false}$  then ...
- To output the contents (e.g., the vertices) of a connected (strongly connected) graph
  - What if the graph is not connected (weakly connected)? Add just a little bit of code and invoke method  $BFS(s) \Rightarrow$  discussed later.

25

## Other Applications of BFS

- To find the shortest path from a vertex  $s$  to a vertex  $v$  in an unweighted graph
- To find the length of such a path
- To find out if a graph contains cycles
- To find the connected components of a graph that is not connected
- To construct a BSF tree/forest from a graph

26



Next time ...

- Depth First Search (DFS)
- Review — Dec. 8
- Final exam — Dec. 11