Data Structures and Algorithms

Lecture 15:

Graphs and BFS II

Department of Computer Science & Technology
United International College

Shortest Path Recording

- BFS we saw only tells us whether a path exists from source s, to other vertices v.
  - It doesn't tell us the path!
  - We need to modify the algorithm to record the path

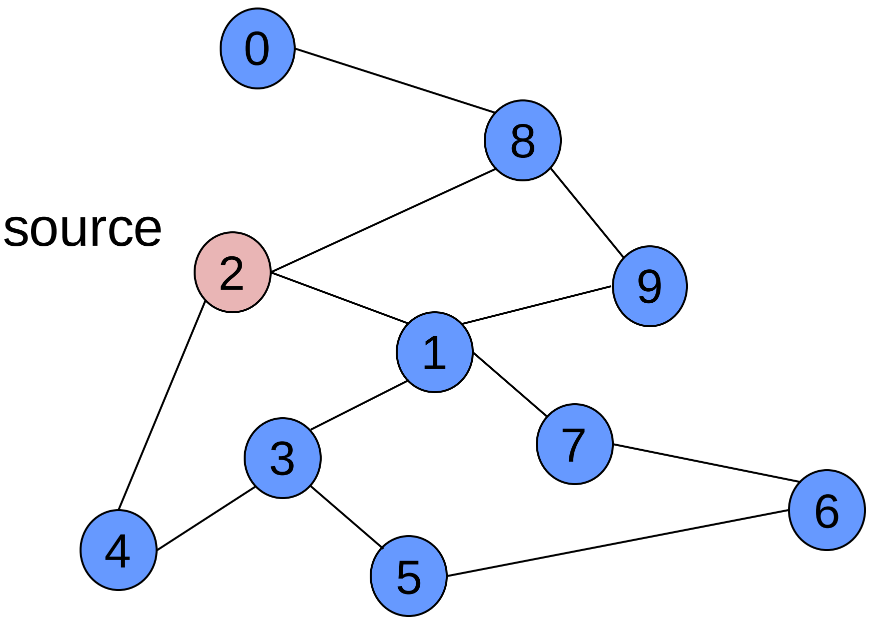
- How can we do that?
  - Note: we do not know which vertices lie on this path until we reach v!
  - Efficient solution:
    - Use an additional array *pred[0..n-1]*
    - *Pred[w] = v means that vertex w was visited from v*

BFS + Path Finding

Algorithm *BFS*(s)

for each vertex v
    do $\mathit{flag}(v) :=$ false;
        $\boxed{\mathit{pred}[v] := -1;}$ initialize all pred[v] to -1
$Q =$ empty queue;
$\mathit{flag}[s] :=$ true;
enqueue(Q, s);
while Q is not empty
    do $v :=$ *dequeue*(Q);
        for each w adjacent to v
            do if $\mathit{flag}[w] =$ false
                then $\mathit{flag}[w] :=$ true;
                    $\boxed{\mathit{pred}[w] := v;}$ Record where you came from
                    enqueue(Q, w)

Example



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)

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

Pred

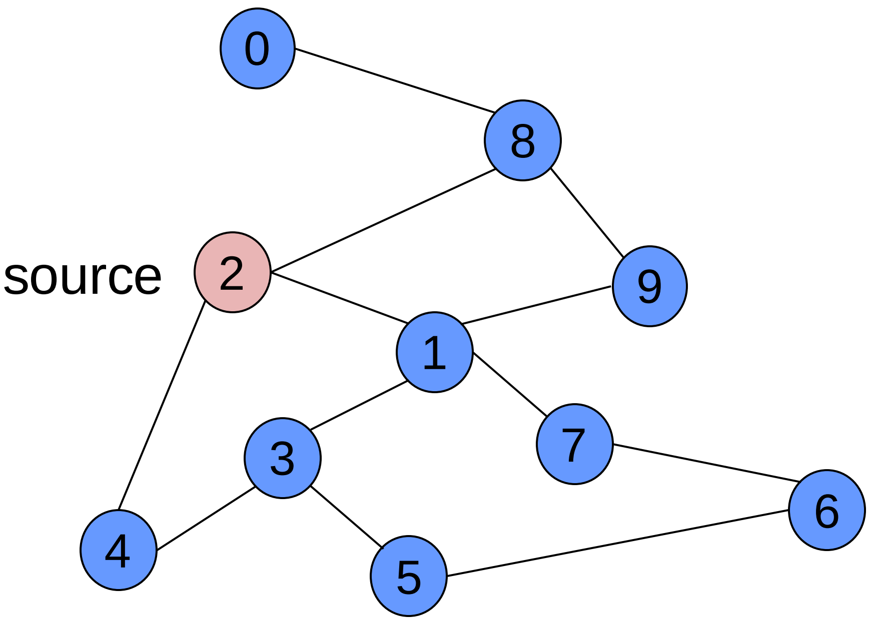
Q = { }

Initialize **Q** to be empty

Initialize visited table (all False)

Initialize Pred to -1

Example (Cont'd)



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)

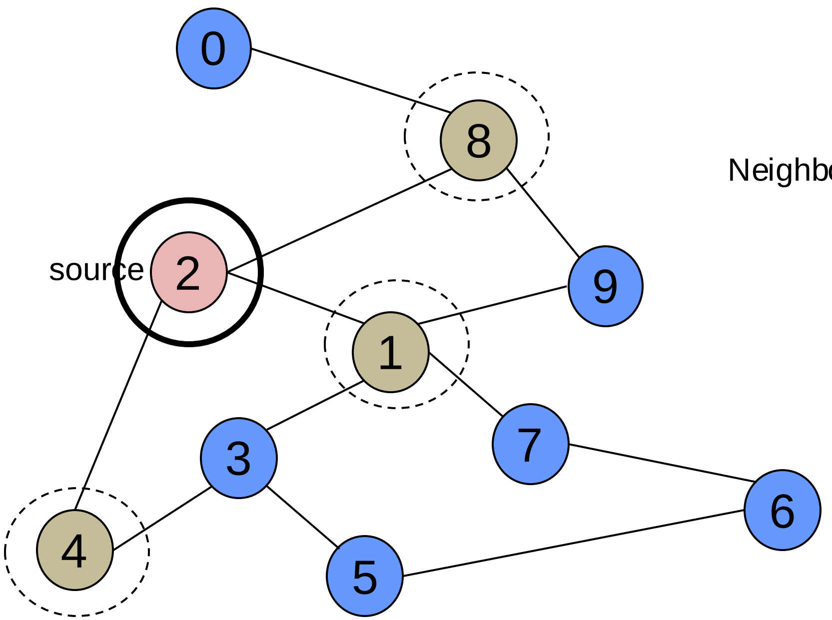
		Pred
0	F	-
1	F	-
2	T	-
3	F	-
4	F	-
5	F	-
6	F	-
7	F	-
8	F	-
9	F	-

Flag that 2 has been visited.

Q = { 2 }

Place source 2 on the queue.

Example (Cont'd)



Adjacency List

	Neighbors
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)

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

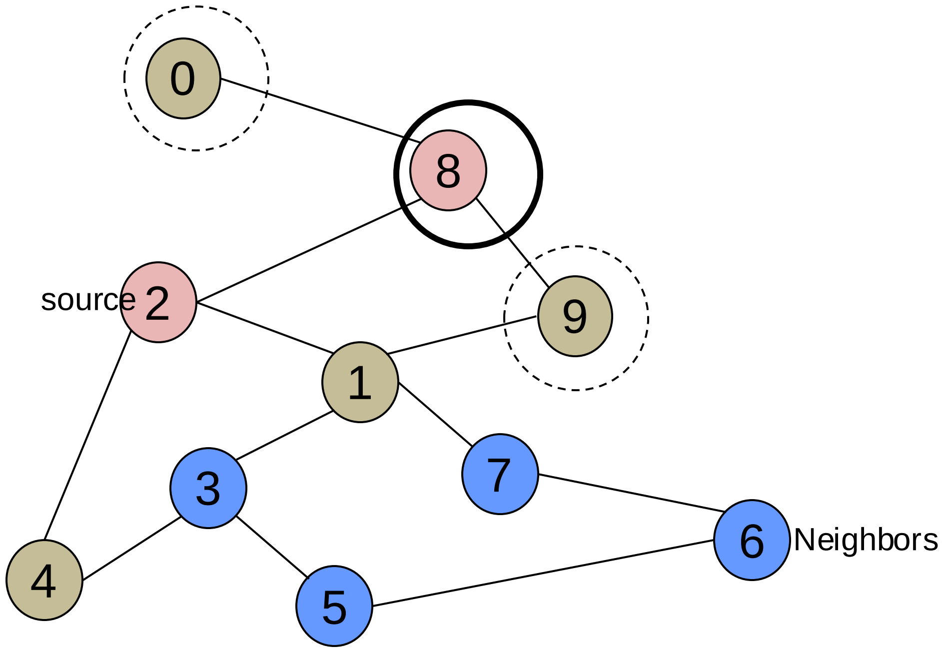
Mark neighbors as visited.

Q = { 2 } → { 8, 1, 4 }

Record in Pred that we came from 2.

Dequeue 2.
Place all unvisited neighbors of 2 on the queue

Example (Cont' d)



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)

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

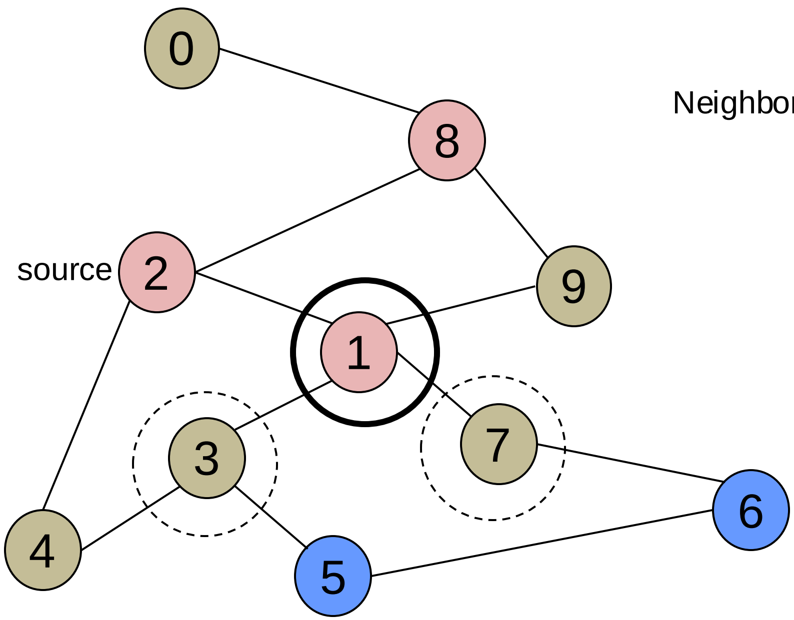
Q = { 8, 1, 4 } → { 1, 4, 0, 9 }

Mark new visited neighbors.

Record in Pred that we came from 8.

Dequeue 8.

- Place all unvisited neighbors of 8 on the queue.
- Notice that 2 is not placed on the queue again, it has been visited!



Adjacency List

Neighbors →

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)

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

Mark new visited neighbors.

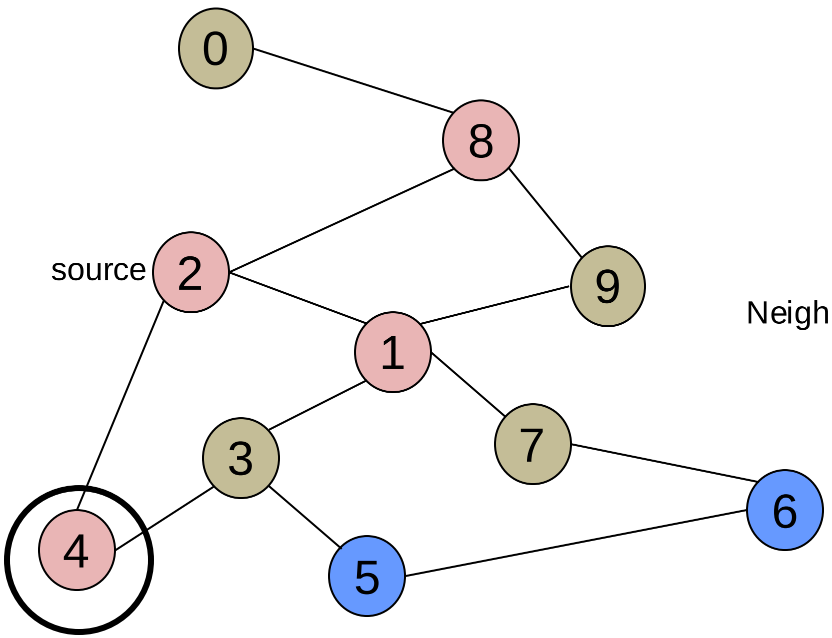
Record in Pred that we came from 1.

Q = { 1, 4, 0, 9 } → { 4, 0, 9, 3, 7 }

Dequeue 1.

- Place all unvisited neighbors of 1 on the queue.
- Only nodes 3 and 7 haven't been visited yet.

Example (Cont’d)



Adjacency List

Neighbors → 4

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)

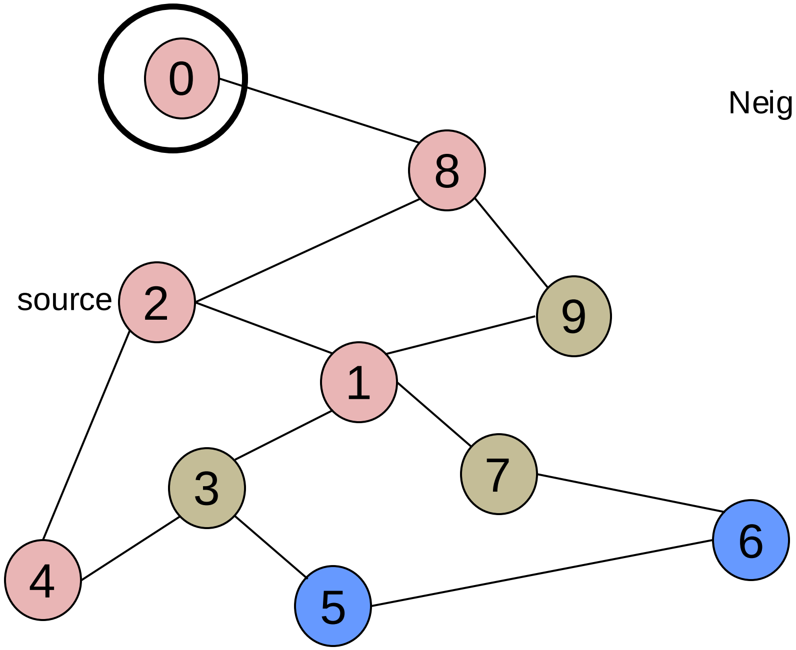
		Pred
0	T	8
1	T	2
2	T	-
3	T	1
4	T	2
5	F	-
6	F	-
7	T	1
8	T	2
9	T	8

Q = { 4, 0, 9, 3, 7 } → { 0, 9, 3, 7 }

Dequeue 4.

- 4 has no unvisited neighbors!

Example (Cont' d)



Adjacency List

Neighbors →

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)

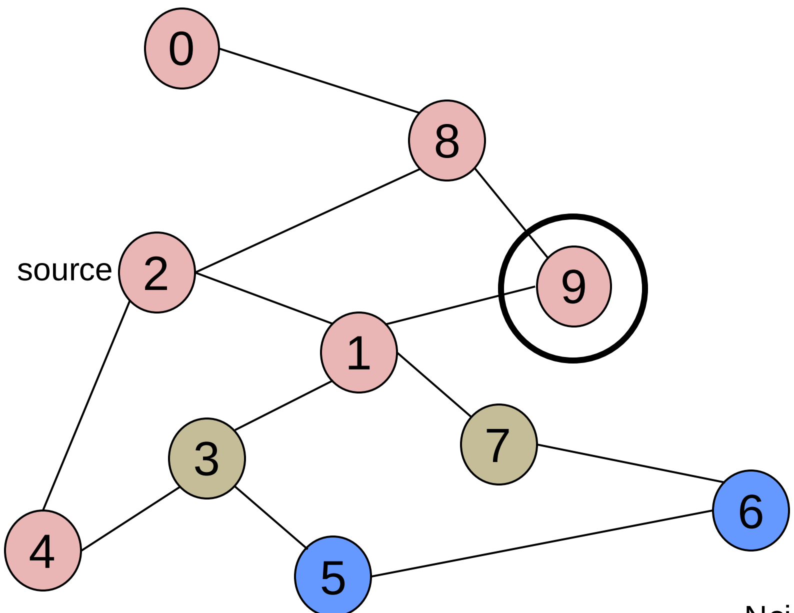
		Pred
0	T	8
1	T	2
2	T	-
3	T	1
4	T	2
5	F	-
6	F	-
7	T	1
8	T	2
9	T	8

Q = { 0, 9, 3, 7 } → { 9, 3, 7 }

Dequeue 0.

- 0 has no unvisited neighbors!

Example (Cont’d)



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

Neighbors → 9

Visited Table (T/F)

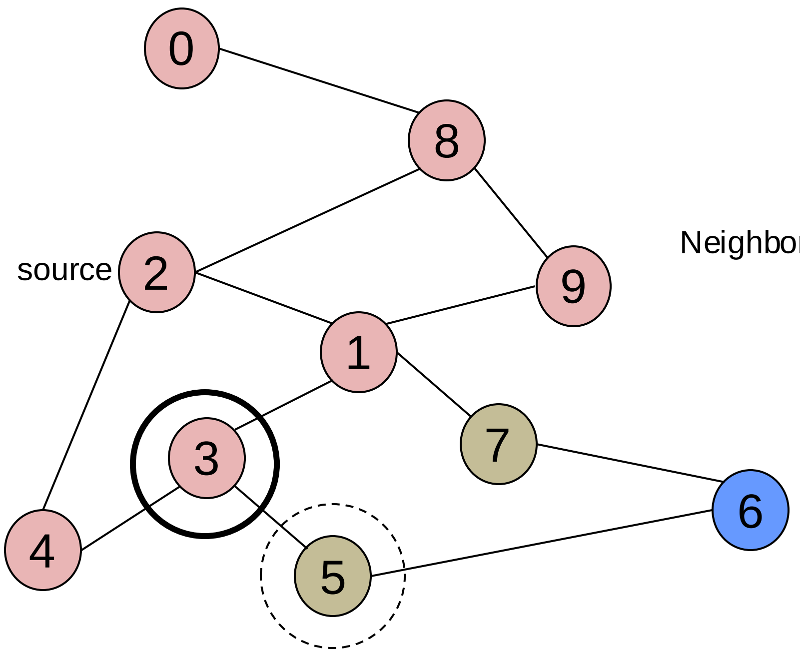
		Pred
0	T	8
1	T	2
2	T	-
3	T	1
4	T	2
5	F	-
6	F	-
7	T	1
8	T	2
9	T	8

Q = { 9, 3, 7 } → { 3, 7 }

Dequeue 9.

- 9 has no unvisited neighbors!

Example (Cont’ d)



Adjacency List

Neighbors →

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)

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

Mark new visited vertex 5.

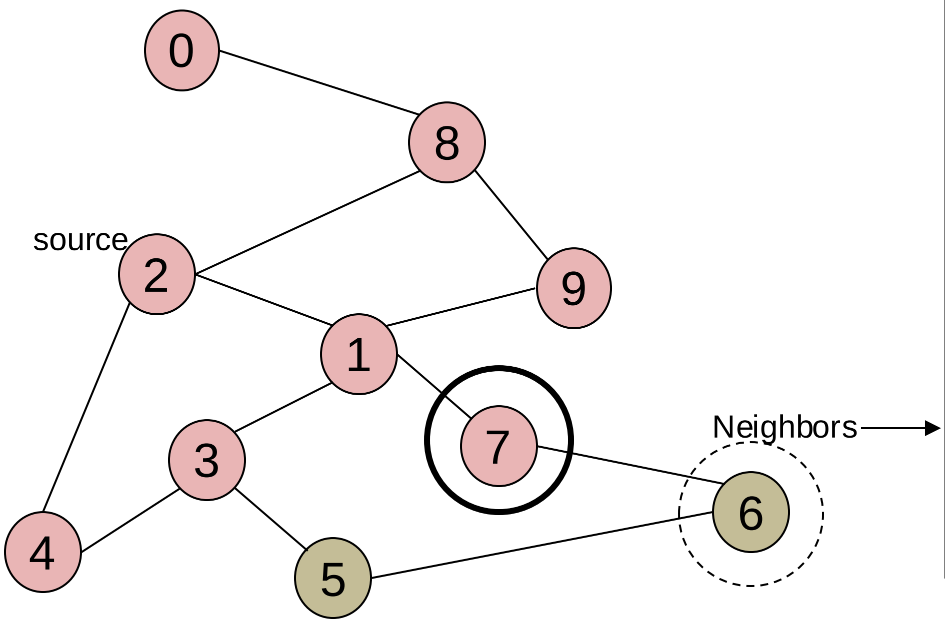
Record in Pred that we came from 3.

Q = { 3, 7 } → { 7, 5 }

Dequeue 3.

- place neighbor 5 on the queue.

Example (Cont' d)



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)

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

Mark new visited vertex 6.

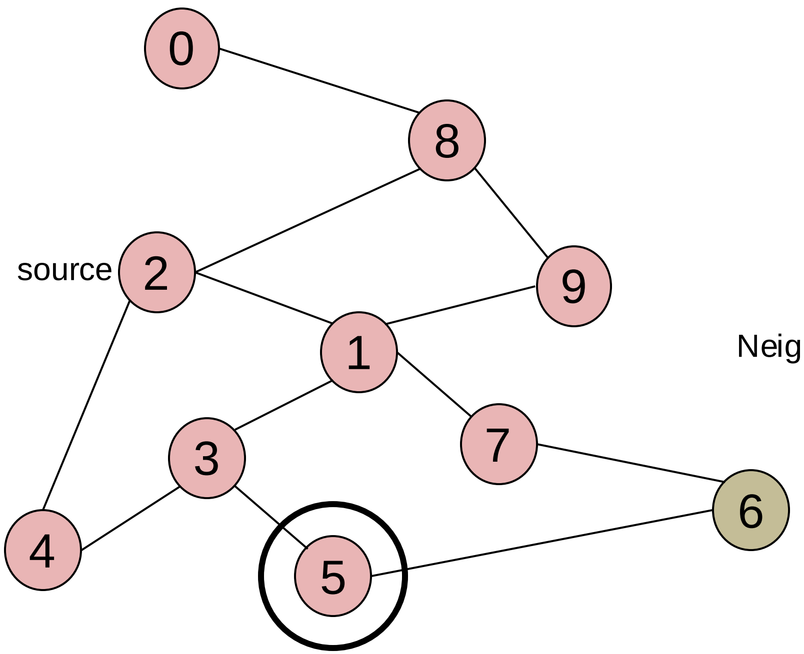
Record in Pred that we came from 7.

Q = { 7, 5 } → { 5, 6 }

Dequeue 7.

- place neighbor 6 on the queue.

Example (Cont’d)



Neighbors →

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)

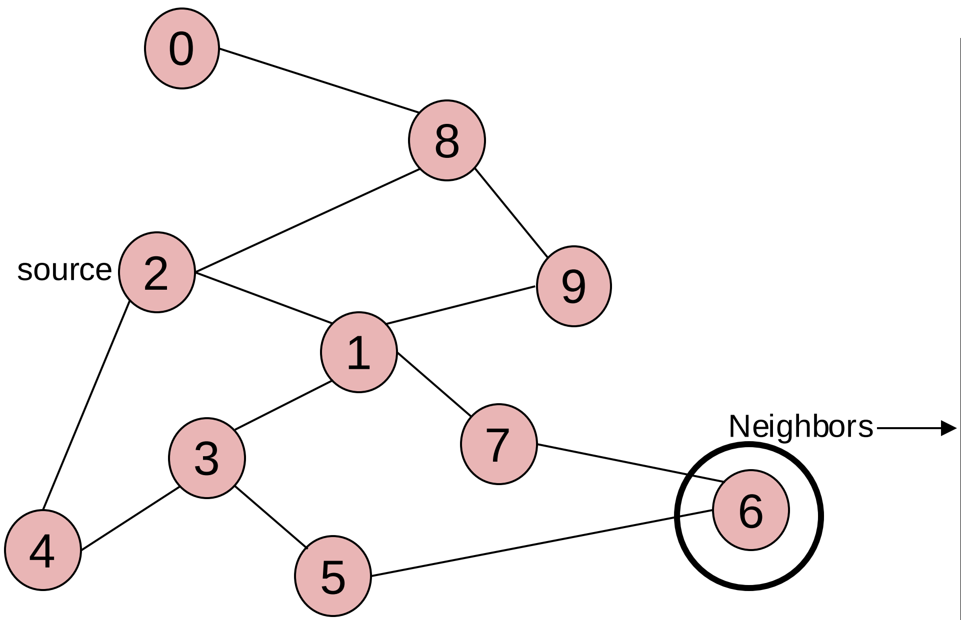
		Pred
0	T	8
1	T	2
2	T	-
3	T	1
4	T	2
5	T	3
6	T	7
7	T	1
8	T	2
9	T	8

Q = { 5, 6} → { 6 }

Dequeue 5.

- no unvisited neighbors of 5.

Example (Cont’d)



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)

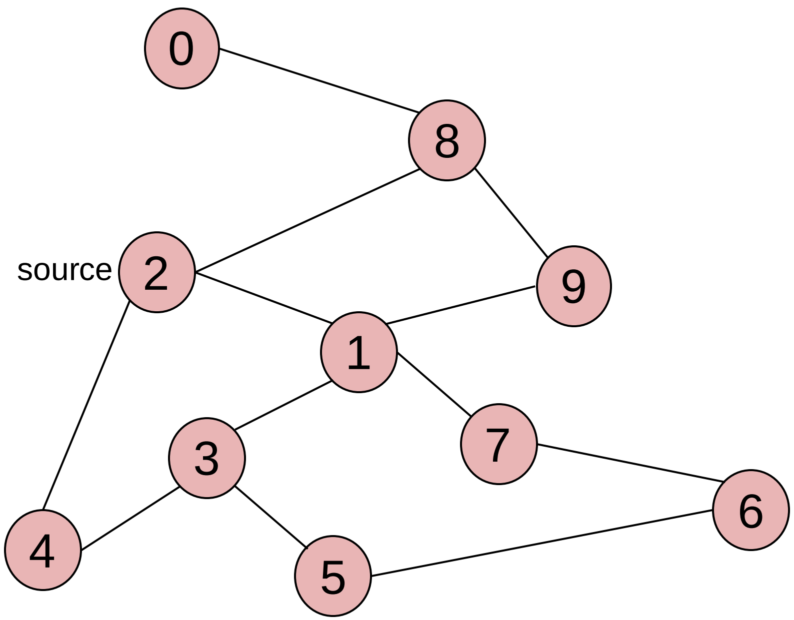
		Pred
0	T	8
1	T	2
2	T	-
3	T	1
4	T	2
5	T	3
6	T	7
7	T	1
8	T	2
9	T	8

Q = { 6 } → { }

Dequeue 6.

- no unvisited neighbors of 6.

BFS Finished



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)

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

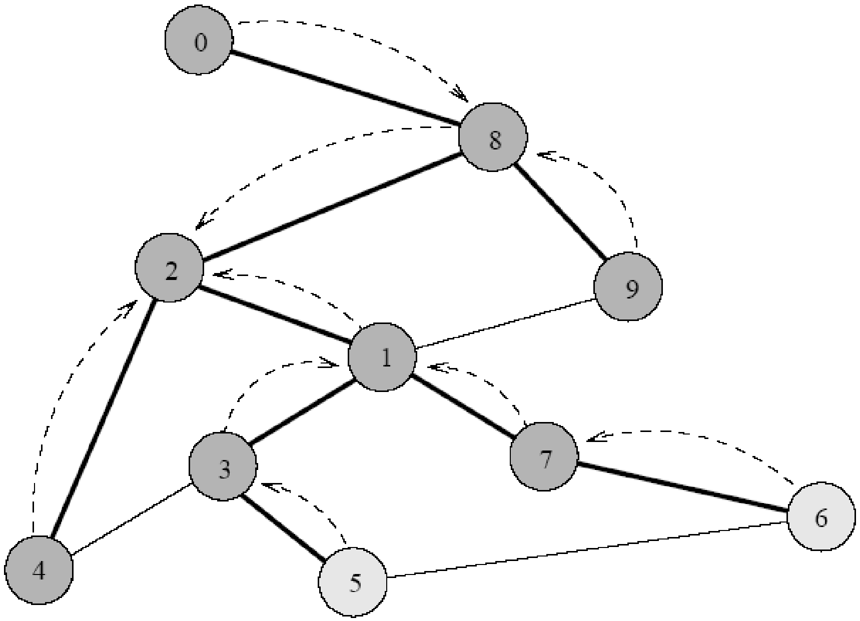
Pred

Q = { }

STOP!!! Q is empty!!!

Pred now can be traced backward to report the path!

Path Reporting



nodes	visited from
0	8
1	2
2	-
3	1
4	2
5	3
6	7
7	1
8	2
9	8

Recursive algorithm

Algorithm *Path*(w)
1. **if** $pred[w] \neq -1$
2. **then**
3. *Path*$(pred[w])$;
4. output w

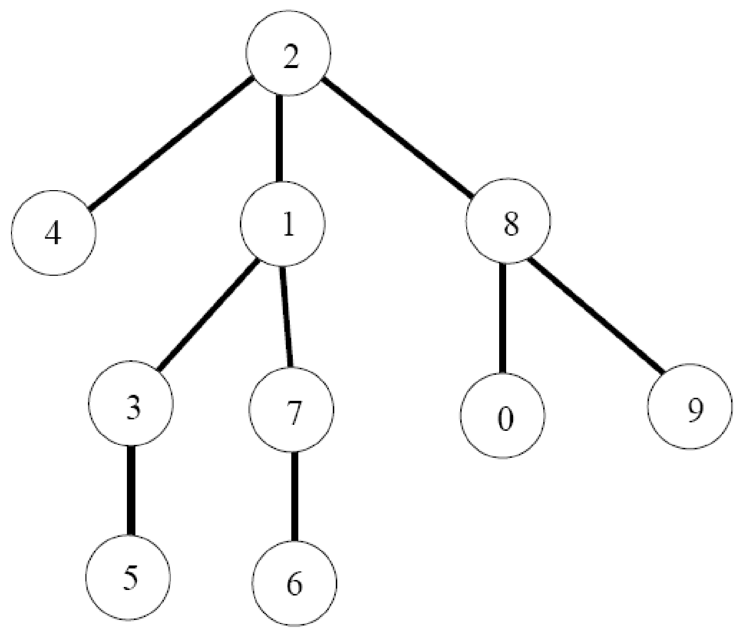
Try some examples, report path from s to v:
Path(0) ->
Path(6) ->
Path(1) ->

The path returned is the shortest from s to v (minimum number of edges).

BFS Tree

- The paths found by BFS is often drawn as a rooted tree (called BFS tree), with the starting vertex as the root of the tree.

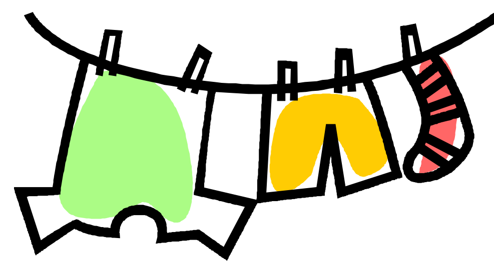
BFS tree for vertex s = 2



Question: What would a “level” order traversal tell you?

Record the Shortest Distance

```
Algorithm BFS(s)
    for each vertex v
        do flag(v) := false;
           pred[v] := −1;  d(v) = ∞;
    Q = empty queue;
    flag[s] := true;      d(s) = 0;
    enqueue(Q, s);
    while Q is not empty
        do v := dequeue(Q);
           for each w adjacent to v
               do if flag[w] = false
                      then flag[w] := true;
           d(w)=d(v)+1;    pred[w] := v;
                           enqueue(Q, w)
```



Application of BFS

- One application concerns how to find connected components in a graph

- If a graph has more than one connected components, BFS builds a BFS-forest (not just BFS-tree)!
  - Each tree in the forest is a connected component.