COMP2010
Data Structures and Algorithms

Lecture 16: Depth-First Search

Department of Computer Science & Technology

United International College

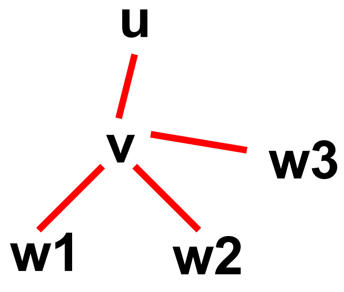
Depth-First Search (DFS)

- DFS is another popular graph search strategy
  - Idea is similar to pre-order traversal (visit node, then visit children recursively)

- DFS can provide certain information about the graph that BFS cannot
  - It can tell whether we have encountered a cycle or not

DFS Algorithm

- DFS will continue to visit neighbors in a recursive pattern
  - Whenever we visit v from u, we recursively visit all unvisited neighbors of v. Then we backtrack (return) to u.
  - Note: it is possible that w2 was unvisited when we recursively visit w1, but became visited by the time we return from the recursive call.



DFS Algorithm

Algorithm *DFS*(s)
1. **for** each vertex v
2. **do** $\boxed{flag[v] := \text{false};}$
3. *RDFS*(s);

Flag all vertices as not visited

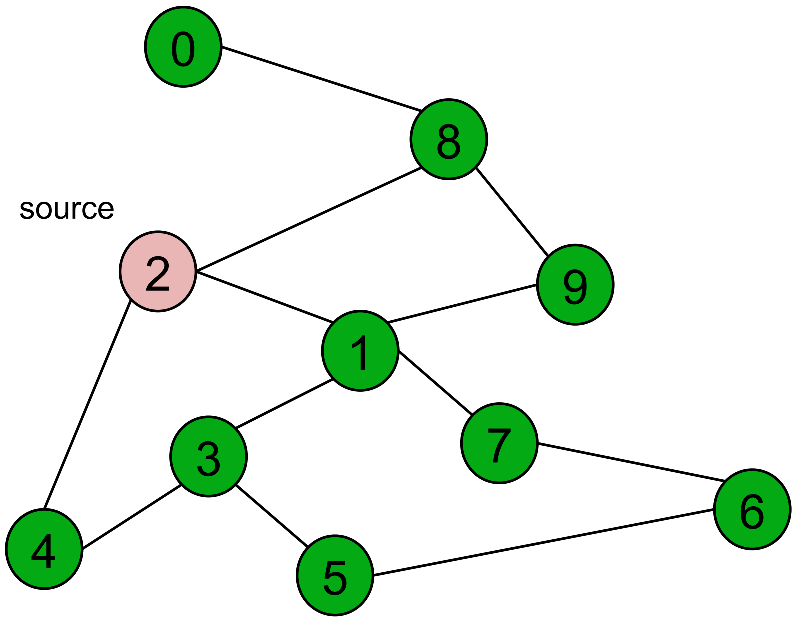
Algorithm *RDFS*(v)
1. $\boxed{flag[v] := \text{true};}$
2. **for** each neighbor w of v
3. **do if** $flag[w] = \text{false}$
4. **then** *RDFS*(w);

Flag yourself as visited

For unvisited neighbors, call RDFS(w) recursively

We can also record the paths using pred[ ].

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)

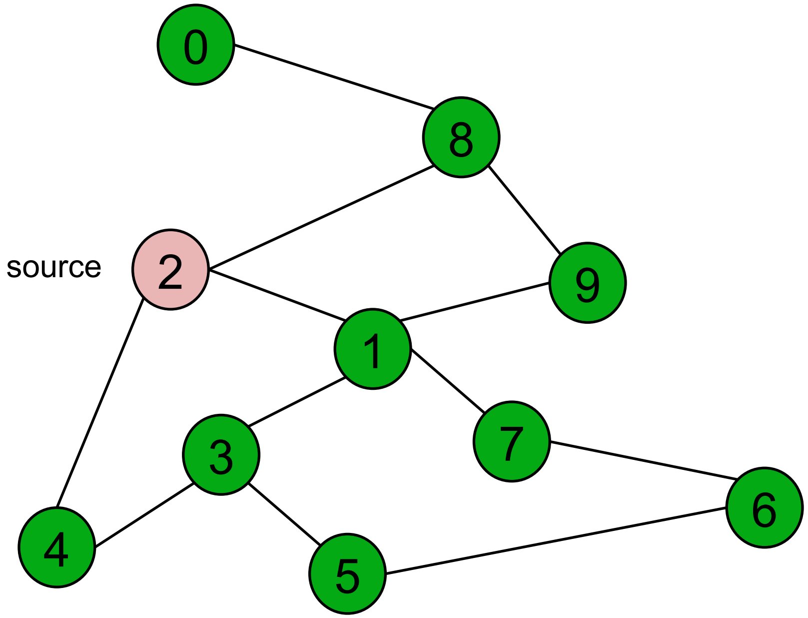
		Pred
0	F	-
1	F	-
2	F	-
3	F	-
4	F	-
5	F	-
6	F	-
7	F	-
8	F	-
9	F	-

Pred

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)

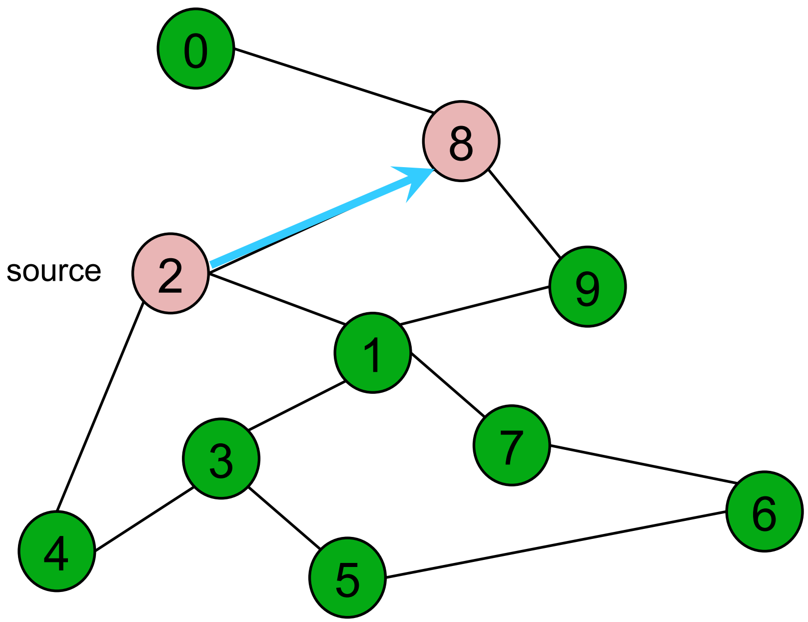
		Pred
0	F	-
1	F	-
2	T	-
3	F	-
4	F	-
5	F	-
6	F	-
7	F	-
8	F	-
9	F	-

Mark 2 as visited

RDFS(2)

Now visit RDFS(8)

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	T	2
9	F	-

Pred

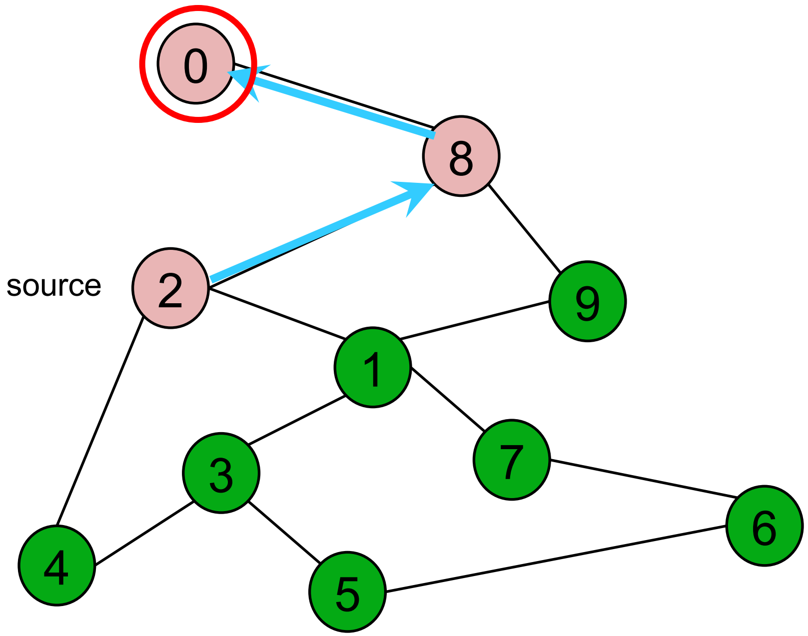
Mark 8 as visited

mark Pred[8]

Recursive calls

RDFS(2)
RDFS(8)
2 is already visited, so visit RDFS(0)

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	F	-
2	T	-
3	F	-
4	F	-
5	F	-
6	F	-
7	F	-
8	T	2
9	F	-

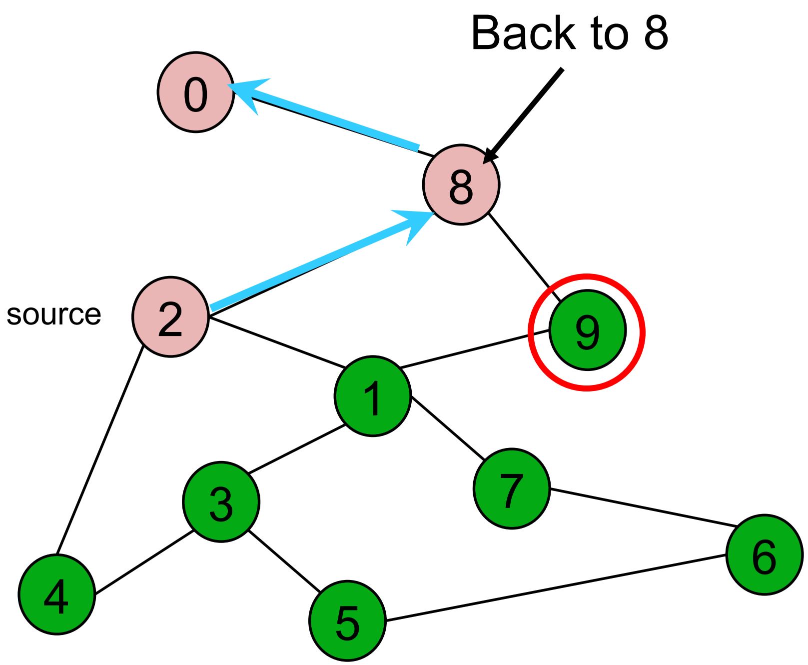
Mark 0 as visited

Mark Pred[0]

Recursive calls

RDFS(2)
  RDFS(8)
    RDFS(0) -> no unvisited neighbors, return to call RDFS(8)

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	F	-
2	T	-
3	F	-
4	F	-
5	F	-
6	F	-
7	F	-
8	T	2
9	F	-

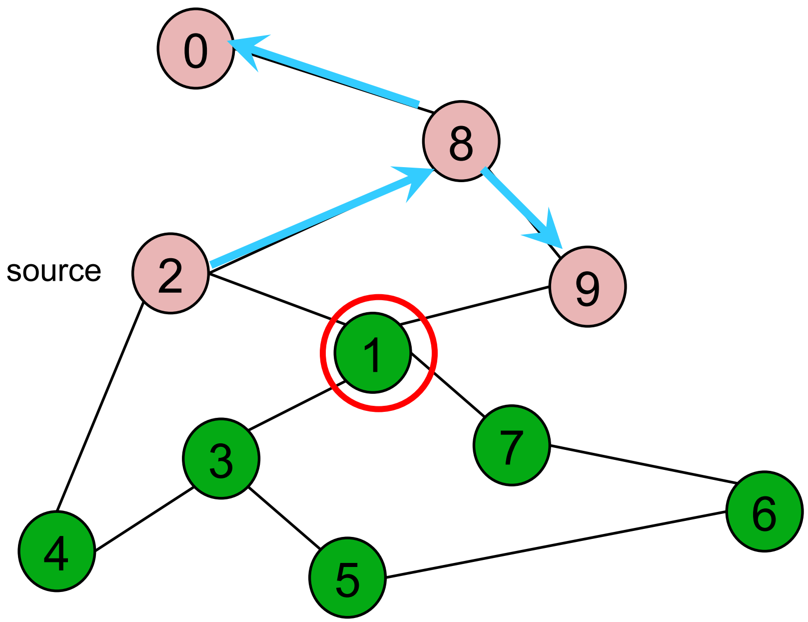
Recursive calls

RDFS(2)

RDFS(8)

Now visit 9 -> RDFS(9)

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	F	-
2	T	-
3	F	-
4	F	-
5	F	-
6	F	-
7	F	-
8	T	2
9	T	8

Pred

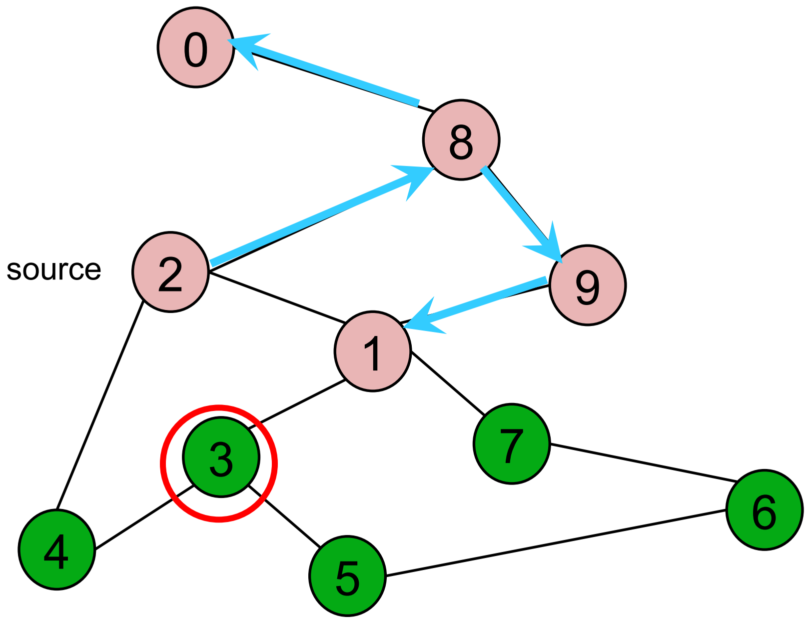
Recursive calls

RDFS(2)
    RDFS(8)
        RDFS(9)
         -> visit 1, RDFS(1)

Mark 9 as visited

Mark Pred[9]

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	9
2	T	-
3	F	-
4	F	-
5	F	-
6	F	-
7	F	-
8	T	2
9	T	8

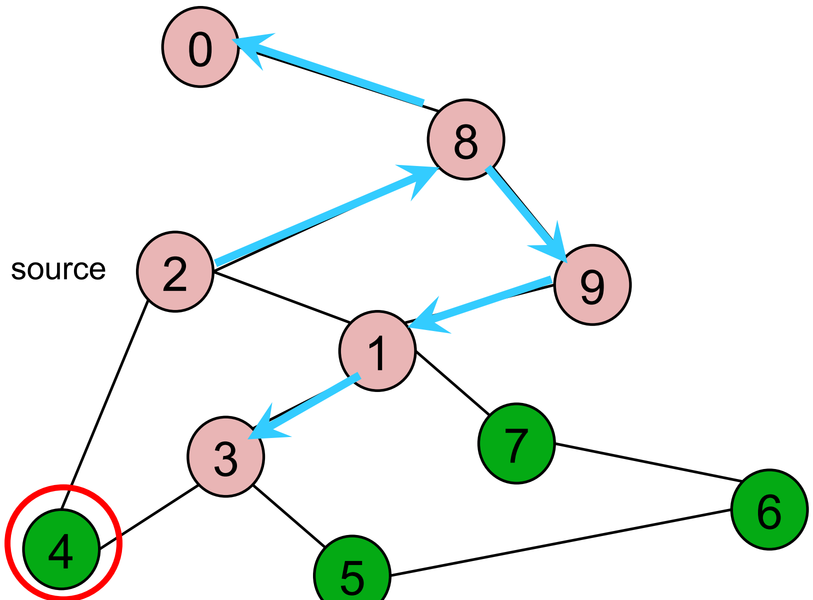
Mark 1 as visited

Mark Pred[1]

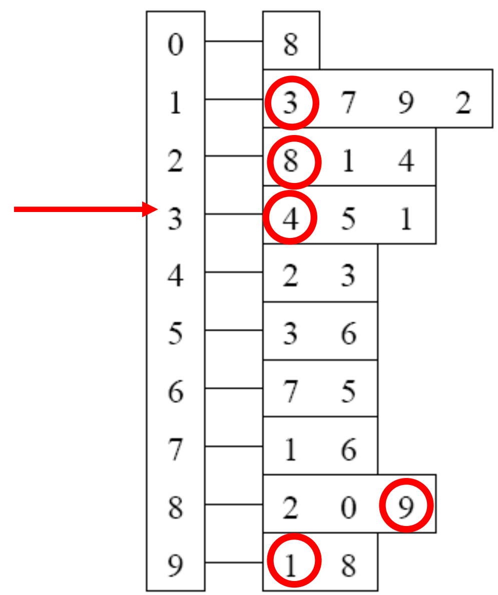
Recursive calls

RDFS(2)
RDFS(8)
RDFS(9)
RDFS(1)
visit RDFS(3)

Example (Cont'd)



Adjacency List



Visited Table (T/F)

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

Pred

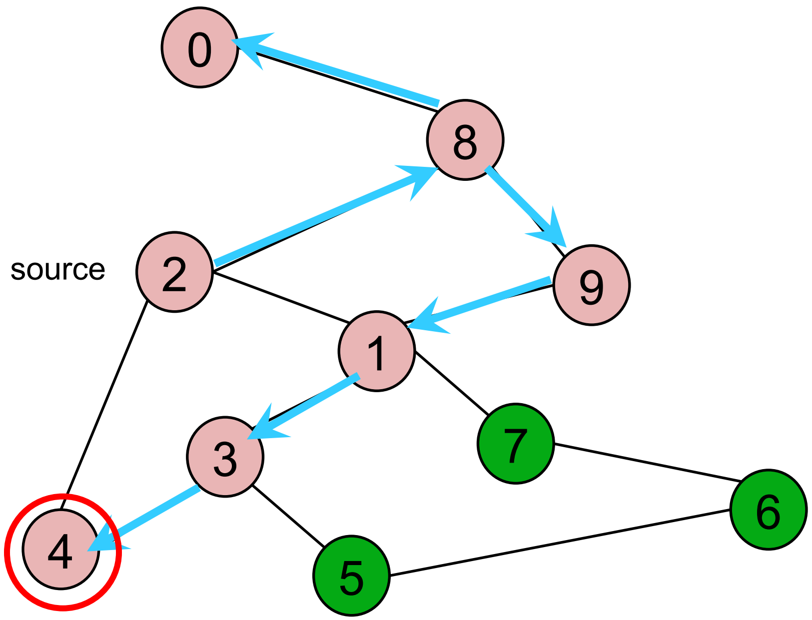
Recursive calls

RDFS(2)
 RDFS(8)
  RDFS(9)
   RDFS(1)
    RDFS(3)
     visit RDFS(4)

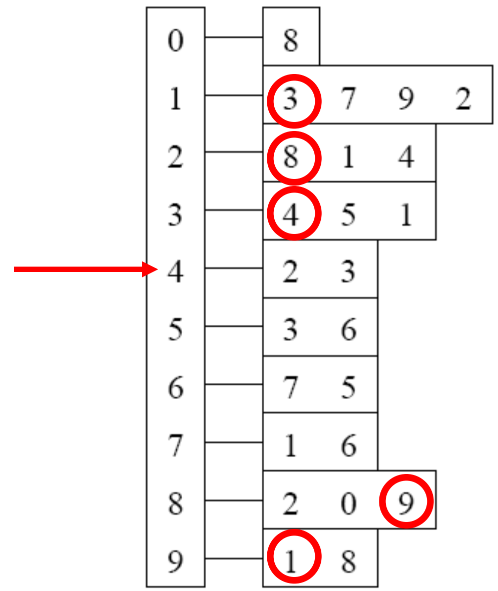
Mark 3 as visited

Mark Pred[3]

Example (Cont'd)



Adjacency List



Visited Table (T/F)

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

Pred

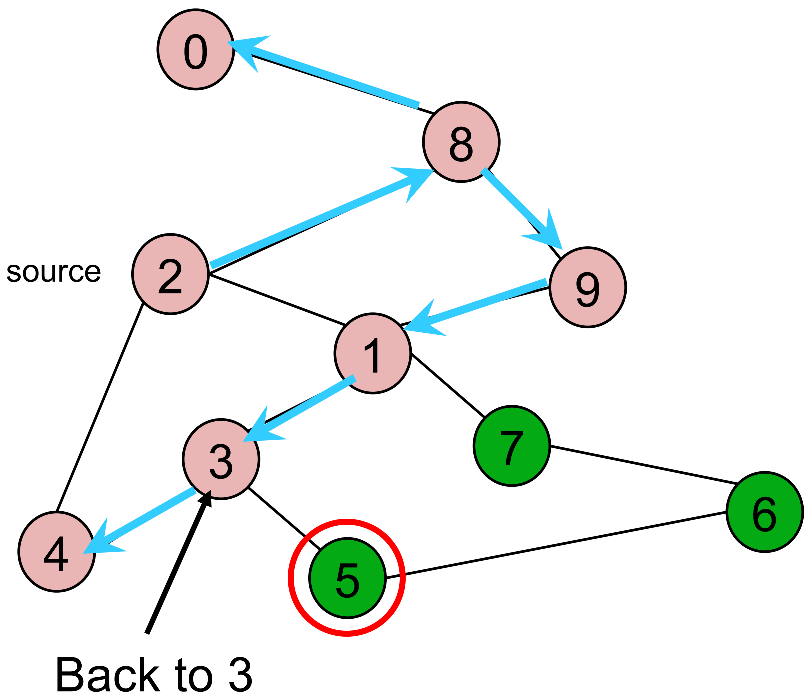
Mark 4 as visited

Mark Pred[4]

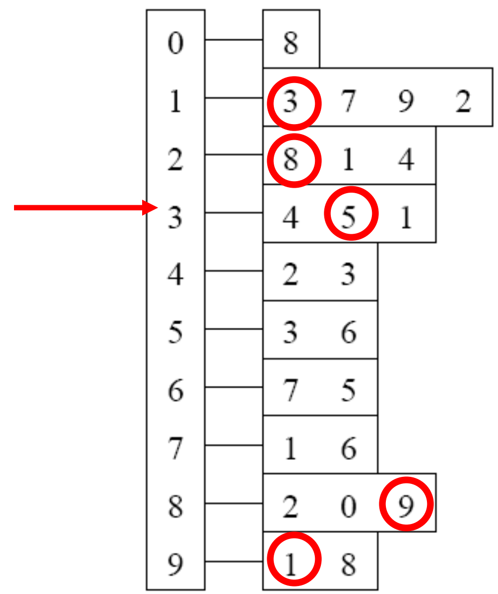
Recursive calls

RDFS(2)
RDFS(8)
RDFS(9)
RDFS(1)
RDFS(3)
RDFS(4) → STOP all of 4's neighbors have been visited
return back to call RDFS(3)

Example (Cont'd)



Adjacency List



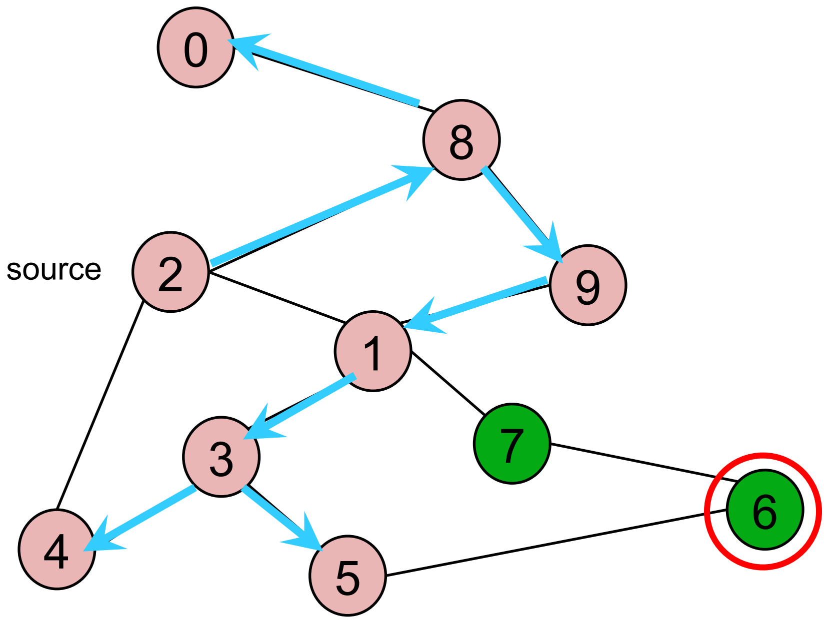
Visited Table (T/F)

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

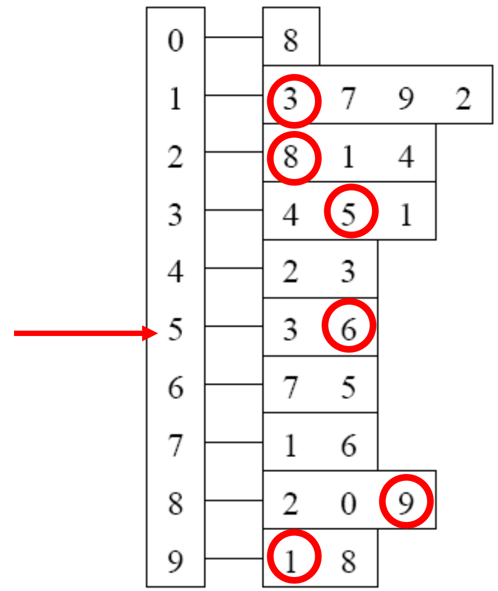
Recursive calls

RDFS(2)
RDFS(8)
RDFS(9)
RDFS(1)
RDFS(3)
visit 5 -> RDFS(5)

Example (Cont'd)



Adjacency List



Visited Table (T/F)

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

Pred

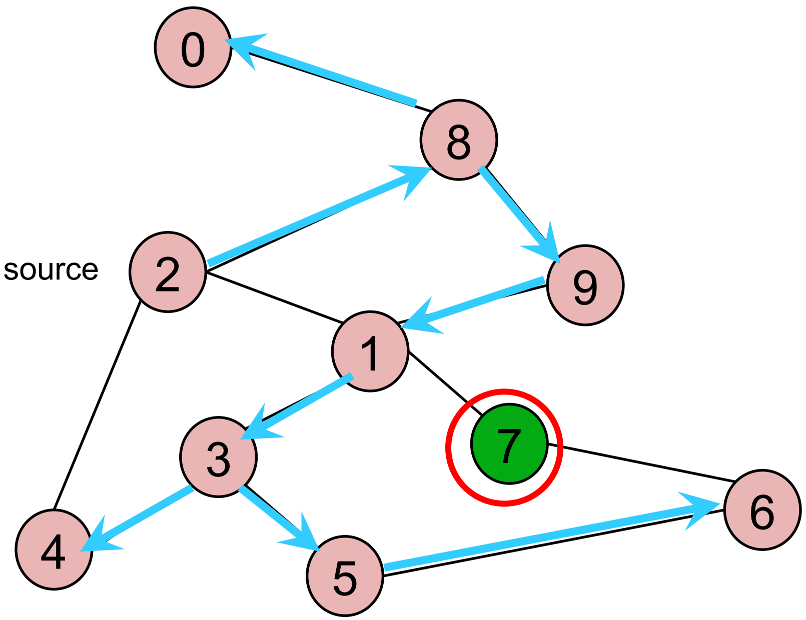
Recursive calls

RDFS(2)
RDFS(8)
RDFS(9)
RDFS(1)
RDFS(3)
RDFS(5)
3 is already visited, so visit 6 -> RDFS(6)

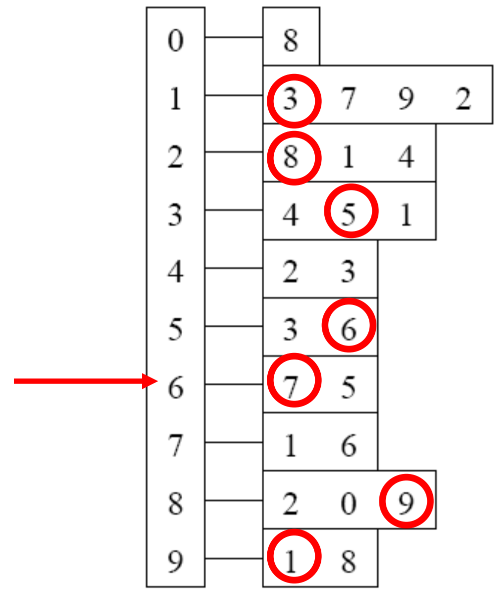
Mark 5 as visited

Mark Pred[5]

Example (Cont'd)



Adjacency List



Visited Table (T/F)

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

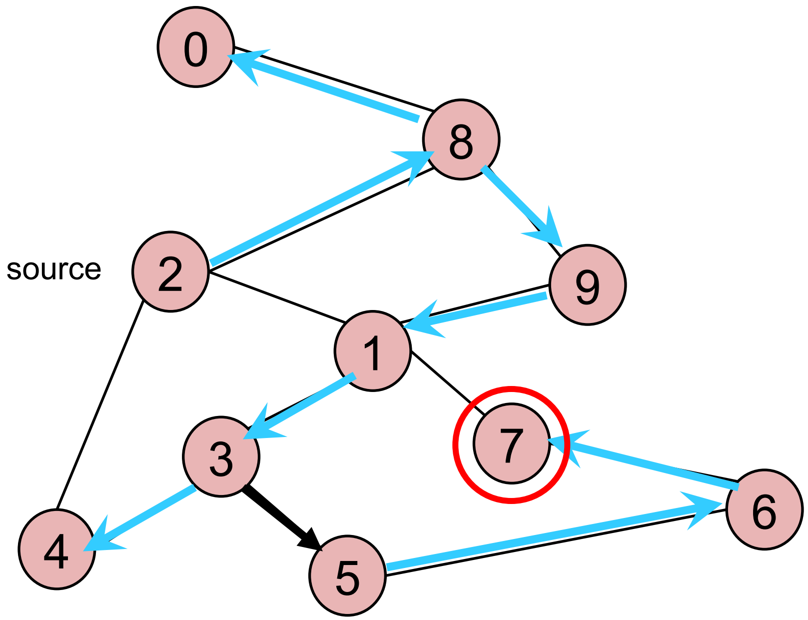
Recursive calls

RDFS(2)
RDFS(8)
RDFS(9)
RDFS(1)
RDFS(3)
RDFS(5)
RDFS(6)
visit 7 -> RDFS(7)

Mark 6 as visited

Mark Pred[6]

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

Pred

Recursive calls

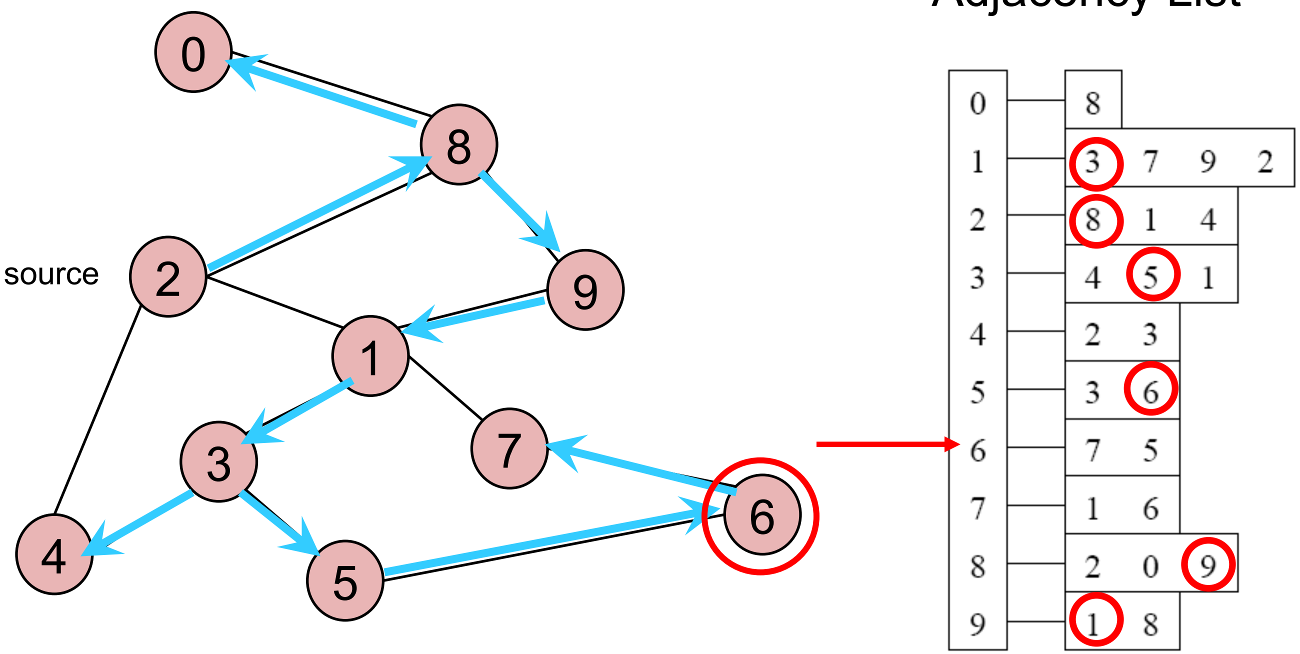
RDFS(2)
RDFS(8)
RDFS(9)
RDFS(1)
RDFS(3)
RDFS(5)
RDFS(6)
RDFS(7) -> Stop no more unvisited neighbors

Mark 7 as visited

Mark Pred[7]

Example (Cont'd)

Adjacency List



Visited Table (T/F)

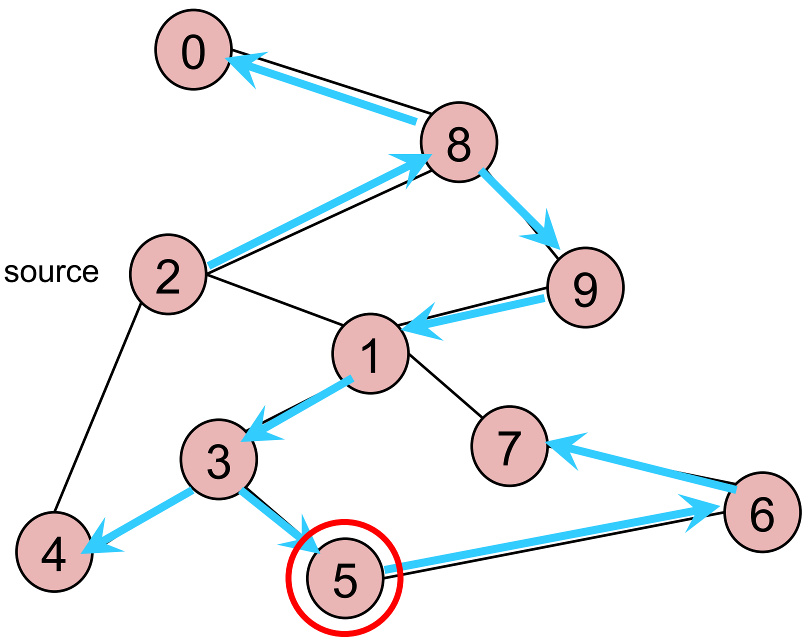
		Pred
0	T	8
1	T	9
2	T	-
3	T	1
4	T	3
5	T	3
6	T	5
7	T	6
8	T	2
9	T	8

Pred

Recursive calls

RDFS(2)
RDFS(8)
RDFS(9)
RDFS(1)
RDFS(3)
RDFS(5)
RDFS(6) -> Stop

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

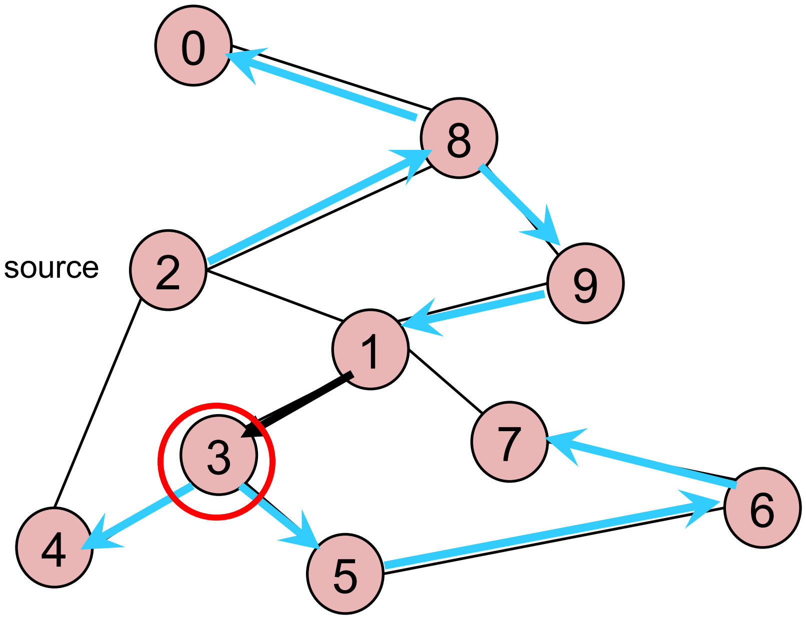
Visited Table (T/F)

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

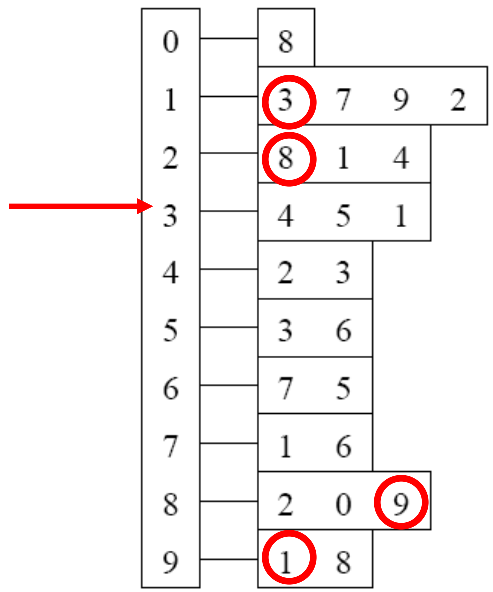
Recursive calls

RDFS(2)
　RDFS(8)
　　RDFS(9)
　　　RDFS(1)
　　　　RDFS(3)
　　　　　RDFS(5) -> Stop

Example (Cont'd)



Adjacency List



Visited Table (T/F)

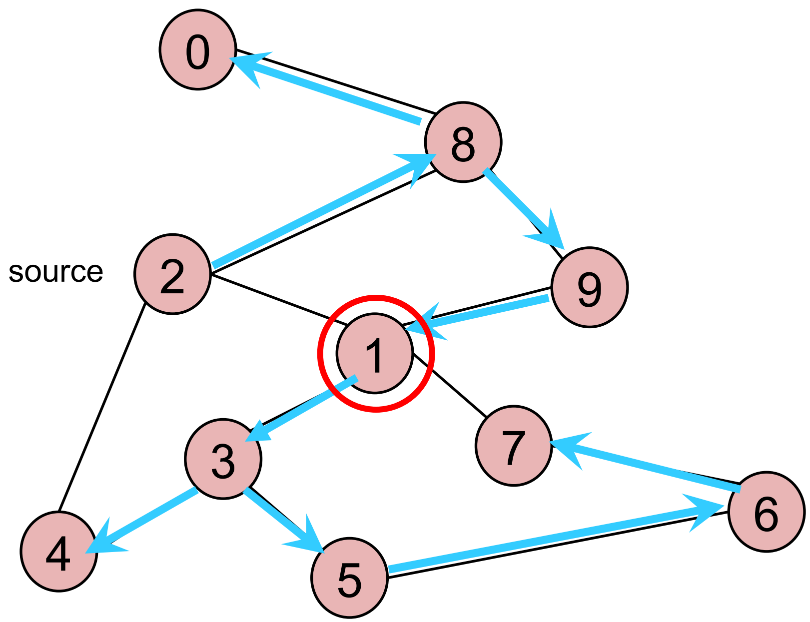
		Pred
0	T	8
1	T	9
2	T	-
3	T	1
4	T	3
5	T	3
6	T	5
7	T	6
8	T	2
9	T	8

Pred

Recursive calls

RDFS(2)
RDFS(8)
RDFS(9)
RDFS(1)
RDFS(3) -> Stop

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		

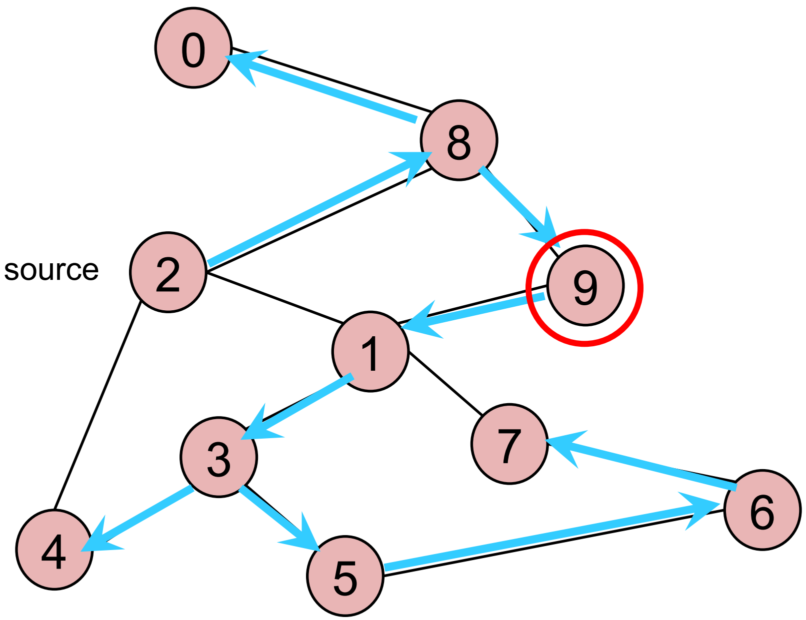
Visited Table (T/F)

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

RDFS(2)
    RDFS(8)
        RDFS(9)
            RDFS(1) -> Stop

Recursive calls

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)

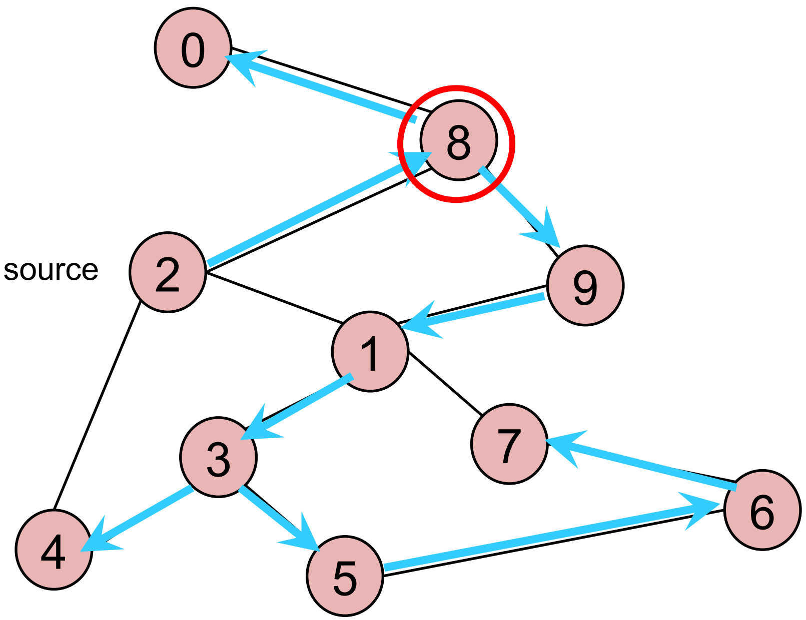
		Pred
0	T	8
1	T	9
2	T	-
3	T	1
4	T	3
5	T	3
6	T	5
7	T	6
8	T	2
9	T	8

Pred

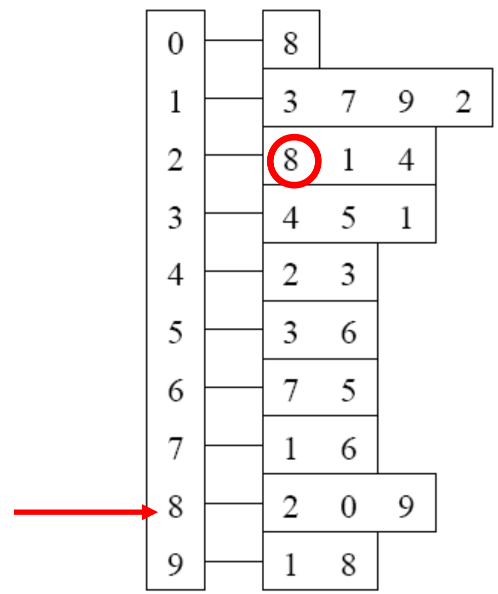
Recursive calls

RDFS(2)
RDFS(8)
RDFS(9) -> Stop

Example (Cont'd)



Adjacency List



Visited Table (T/F)

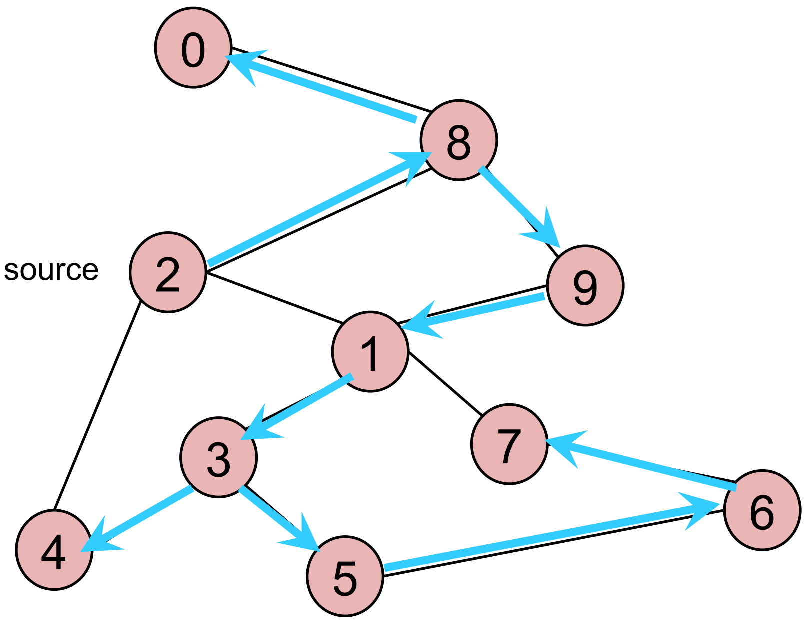
		Pred
0	T	8
1	T	9
2	T	-
3	T	1
4	T	3
5	T	3
6	T	5
7	T	6
8	T	2
9	T	8

Pred

RDFS(2)
RDFS(8) -> Stop

Recursive calls

Example Finished



RDFS(2) -> Stop

Recursive calls finished

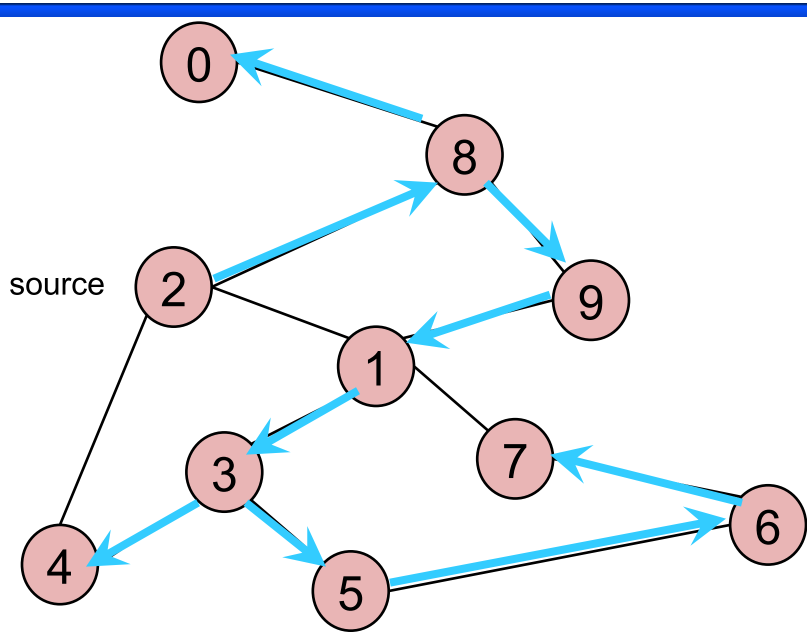
Adjacency List

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

DFS Path Tracking



DFS find out path too

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

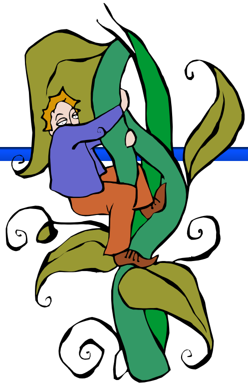
Pred

Algorithm *Path*(w)

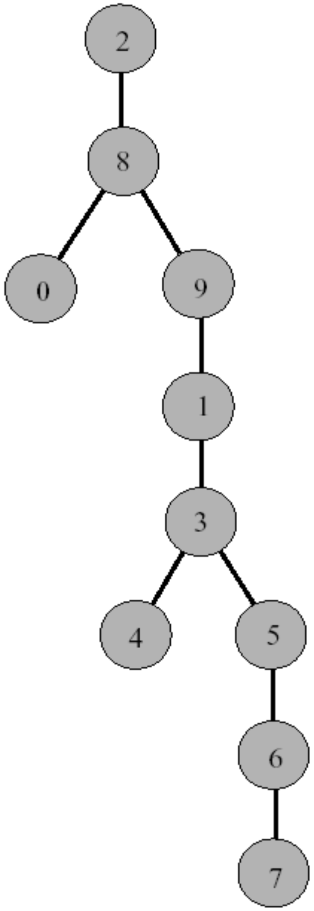
```
if pred[w] ≠ −1
    then
        Path(pred[w]);
output w
```

Try some examples.
Path(0) ->
Path(6) ->
Path(7) ->

DFS Tree



Resulting DFS-tree.
Notice it is much “deeper”
than the BFS tree.



Captures the structure of the recursive calls

- when we visit a neighbor w of v, we add w as child of v
- whenever DFS returns from a vertex v, we climb up in the tree from v to its parent

Time Complexity of DFS
(Using adjacency list)

- We never visited a vertex more than once

- We had to examine all edges of the vertices
  - We know $\Sigma_{vertex\ v}$ degree$(v) = 2m$ where m is the number of edges

- So, the running time of DFS is proportional to the number of edges and number of vertices (same as BFS)
  - O(n + m)

- You will also see this written as:
  - O(|v|+|e|)

|v| = number of vertices (n)
|e| = number of edges (m)