

# Basic Graph Algorithms

G. Philip

Institute of Mathematical Sciences, Chennai

[gphilip@imsc.res.in](mailto:gphilip@imsc.res.in)

January 3, 2009

## Problem: Reachability

- Input: A graph  $G$ , and two vertices  $u, v$  of  $G$ .
- Question: Is there a path in  $G$  from  $u$  to  $v$  ?

Add example figures – one Yes, one NO.



# Problem: Reachability

- Solution: Graph Search Algorithms
  - Breadth- First Search (BFS)
  - Depth-First Search (DFS)
  - ...



# Breadth-First Search

- What is BFS?



# Breadth-First Search

- What is BFS?
- Start by “visiting” vertex  $s$ .
- Visit all vertices adjacent to  $s$ .
- Visit all (as yet unvisited) vertices that are “two edges away” from  $s$ .
- Visit all (as yet unvisited) vertices that are three edges away from  $s$ .
- ...
- Till no unvisited vertex is adjacent to any already visited vertex.



# Breadth-First Search

## Implementation

- How do we ensure that:
  - Vertices are visited strictly in the order of their distances from  $u$ , and
  - No vertex is visited twice?



# Breadth-First Search

## Implementation

- How do we ensure that:
  - Vertices are visited strictly in the order of their distances from  $u$ , and
  - No vertex is visited twice?
- We use:
  - A queue to ensure the order, and
  - Marking (colouring) to prevent multiple visits.



# Breadth-First Search

## Implementation

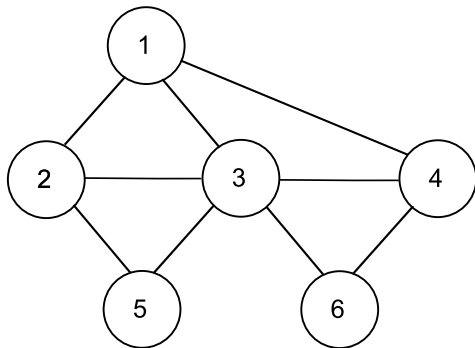
### The Algorithm

- 1 Mark  $s$  as visited. Add  $s$  to the queue.
- 2 Repeat the following till the queue is empty:
  - Remove a vertex (say  $w$ ) from the queue.
  - For each neighbour  $x$  of  $w$  that is not yet visited,
    - Mark  $x$  as visited, and add  $x$  to the queue.



# Breadth-First Search

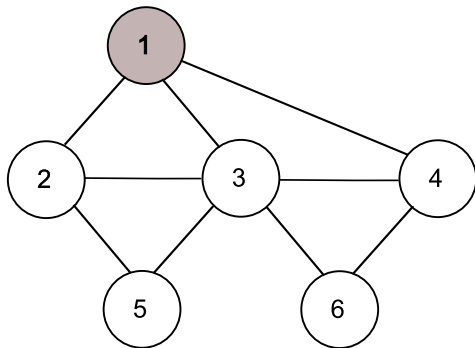
Example



front   
Queue

# Breadth-First Search

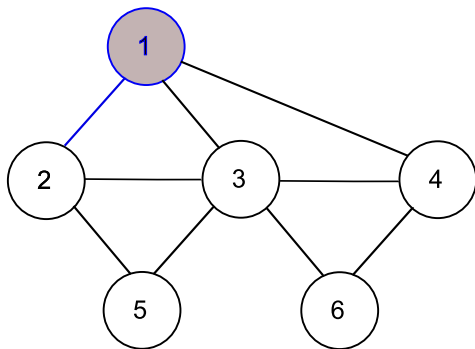
Example



front 1  
Queue

# Breadth-First Search

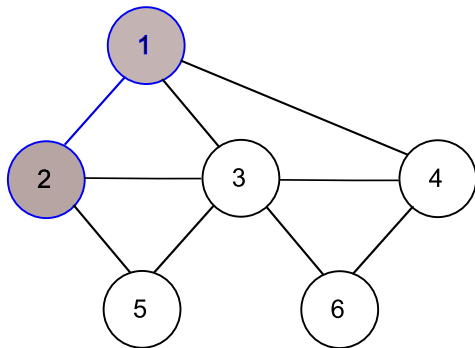
Example



front   
Queue

# Breadth-First Search

Example

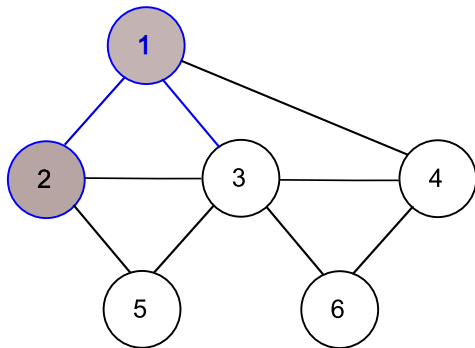


front

Queue

# Breadth-First Search

Example

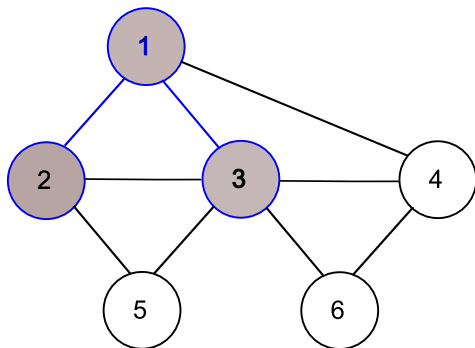


front

Queue

# Breadth-First Search

Example



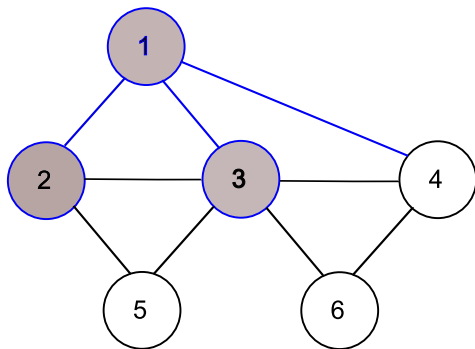
front 

2	3
---	---

  
Queue

# Breadth-First Search

Example



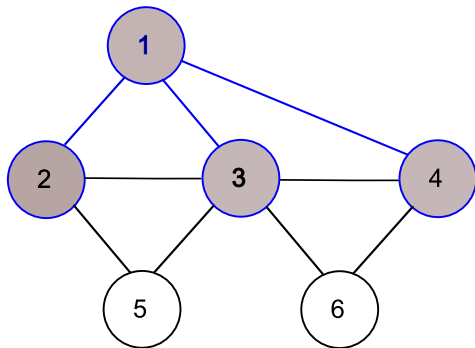
front 

2	3
---	---

  
Queue

# Breadth-First Search

Example



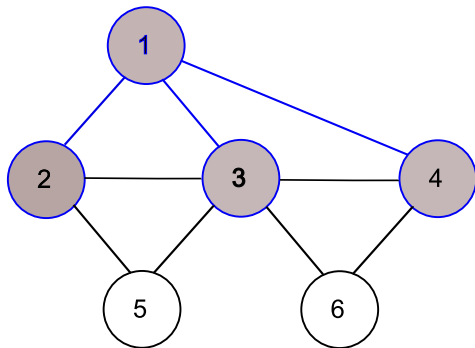
front 

2	3	4
---	---	---

  
Queue

# Breadth-First Search

Example

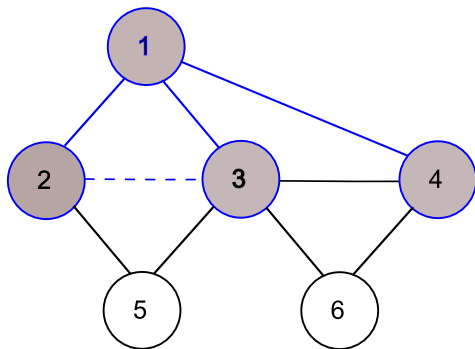


front **3 4**

Queue

# Breadth-First Search

Example

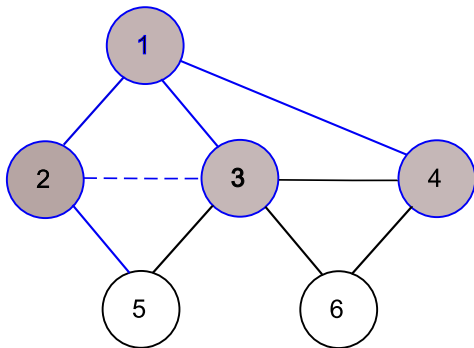


front **3 4**

Queue

# Breadth-First Search

Example

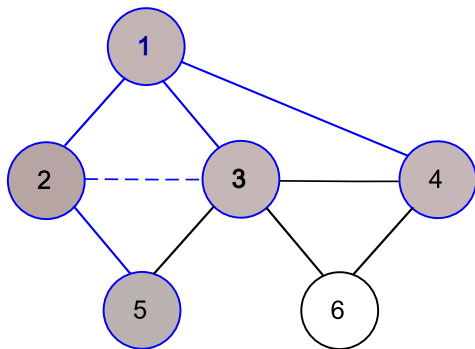


front **3 4**

Queue

# Breadth-First Search

Example

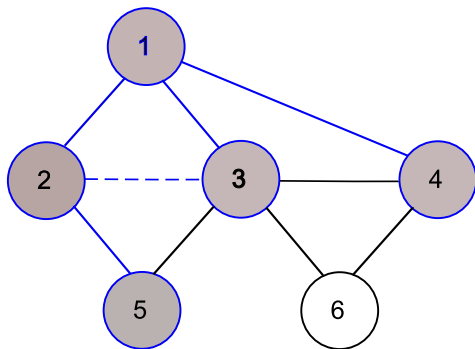


front **3 4 5**

Queue

# Breadth-First Search

Example



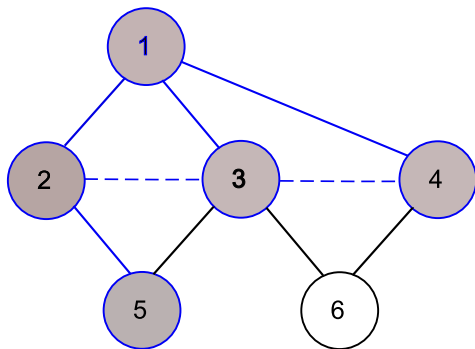
front 

4	5
---	---

  
Queue

# Breadth-First Search

Example



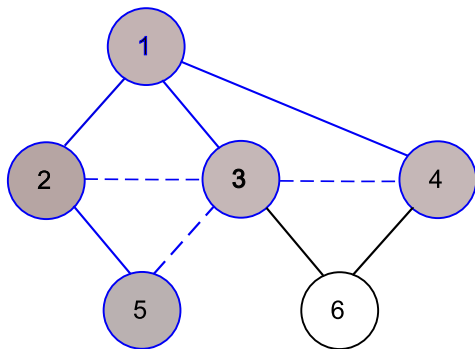
front 

4	5
---	---

  
Queue

# Breadth-First Search

Example



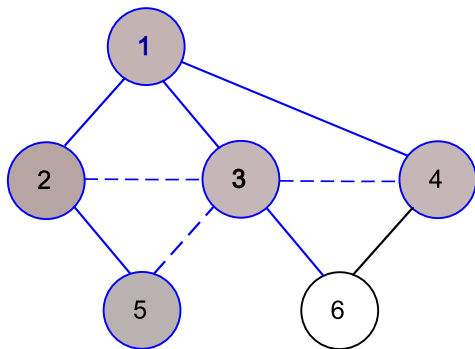
front 

4	5
---	---

  
Queue

# Breadth-First Search

Example



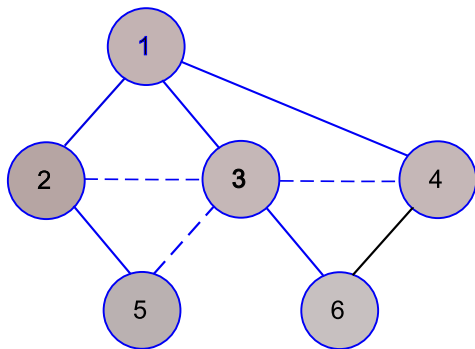
front 

4	5
---	---

  
Queue

# Breadth-First Search

Example



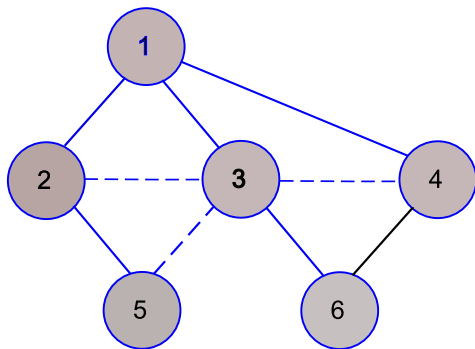
front 

4	5	6
---	---	---

  
Queue

# Breadth-First Search

Example



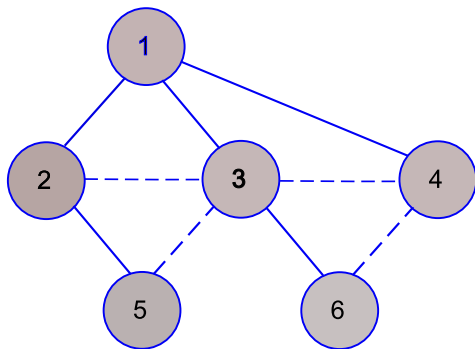
front 

5	6
---	---

  
Queue

# Breadth-First Search

Example



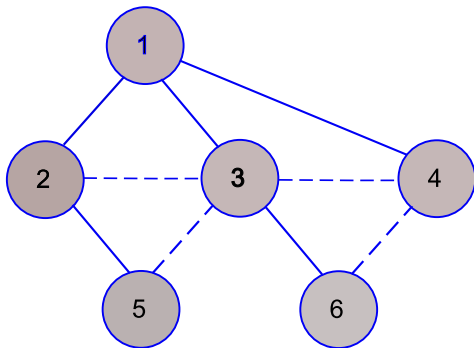
front 

5	6
---	---

  
Queue

# Breadth-First Search

Example

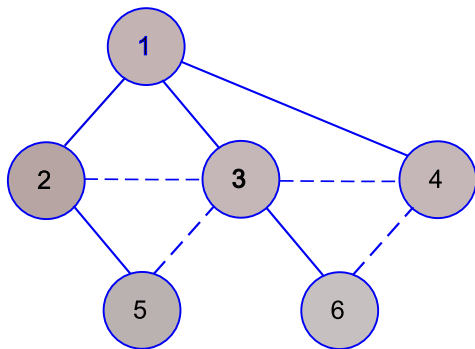


front

Queue

# Breadth-First Search

Example



front

Queue

# Breadth-First Search

## Correctness

- Let  $G$  be the input graph, and suppose we start BFS from vertex  $s$  of  $G$ . Let  $v$  be some vertex of  $G$ .



# Breadth-First Search

## Correctness

- Let  $G$  be the input graph, and suppose we start BFS from vertex  $s$  of  $G$ . Let  $v$  be some vertex of  $G$ .
- Let  $v$  be reachable from  $s$ . In our BFS, will  $v$  ever be visited?



# Breadth-First Search

## Correctness

- Let  $G$  be the input graph, and suppose we start BFS from vertex  $s$  of  $G$ . Let  $v$  be some vertex of  $G$ .
- Let  $v$  be reachable from  $s$ . In our BFS, will  $v$  ever be visited?
- Let  $v$  be **not** reachable from  $s$ . In our BFS, will  $v$  ever be visited?



# Breadth-First Search

## Complexity

### The Algorithm

- 1 Mark  $s$  as visited. Add  $s$  to the queue.
  - 2 Repeat the following till the queue is empty:
    - Remove a vertex (say  $w$ ) from the queue.
    - For each neighbour  $x$  of  $w$  that is not yet visited,
      - Mark  $x$  as visited, and add  $x$  to the queue.
- For a graph with  $n$  vertices and  $m$  edges, how much **space** does this algorithm use?



# Breadth-First Search

## Complexity

### The Algorithm

- 1 Mark  $s$  as visited. Add  $s$  to the queue.
  - 2 Repeat the following till the queue is empty:
    - Remove a vertex (say  $w$ ) from the queue.
    - For each neighbour  $x$  of  $w$  that is not yet visited,
      - Mark  $x$  as visited, and add  $x$  to the queue.
- For a graph with  $n$  vertices and  $m$  edges, how much **time** does this algorithm take?



# Breadth-First Search

## Complexity

Graph with $n$ vertices, $m$ edges	Adjacency Matrix	Adjacency List
Space required	$O(n^2)$	$O(n + m)$
Time required	$O(n^2)$	$O(n + m)$

- Adjacency list – advantageous especially for *sparse* graphs, where  $m \ll n^2$ .



# Depth-First Search

- What is DFS?



# Depth-First Search

- What is DFS?
- Start by “visiting” vertex  $s$ .
- Visit an as yet unvisited neighbour, say  $u_1$ , of  $s$ .
- Visit an as yet unvisited neighbour, say  $u_2$ , of  $u_1$ .
- Visit an as yet unvisited neighbour, say  $u_3$ , of  $u_2$ .
- ...
- If the current vertex  $u_i$  has no unvisited neighbours, then “go up one level” : start the procedure with  $u_{i-1}$ .
- Do this till  $s$  has no unvisited neighbour.

# Depth-First Search

## Implementation

- How do we remember which vertex to “go back to” when the current vertex has no more unvisited neighbours?
- How do we ensure that no vertex is visited twice?



# Depth-First Search

## Implementation

- How do we remember which vertex to “go back to” when the current vertex has no more unvisited neighbours?
- How do we ensure that no vertex is visited twice?
- We use:
  - A stack to remember which vertices to go back to, and
  - Marking (colouring) to prevent multiple visits.



# Depth-First Search

## Implementation

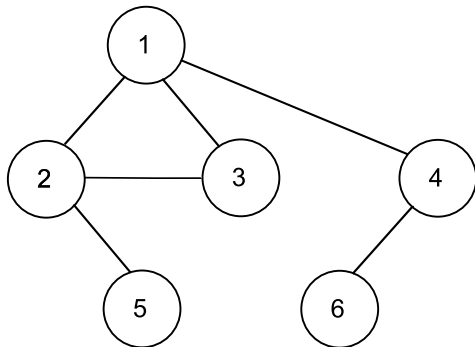
### The Algorithm

- 1 Mark  $s$  as visited. Push  $s$  on to the stack.
- 2 Repeat the following till the stack is empty:
  - Pop a vertex (say  $w$ ) from the stack.
  - For each neighbour  $x$  of  $w$  that is not yet visited,
    - Mark  $x$  as visited, and push  $x$  on to the stack.



# Depth-First Search

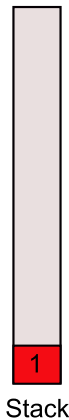
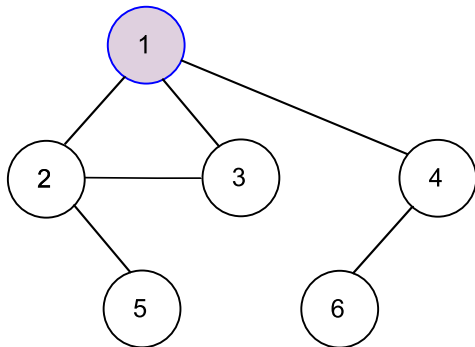
Example



Stack

# Depth-First Search

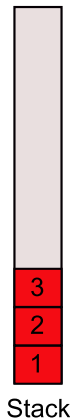
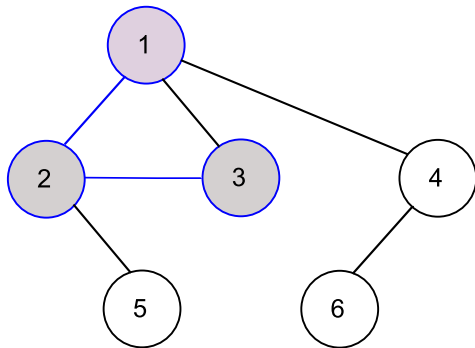
Example





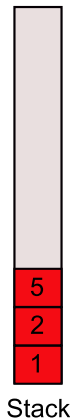
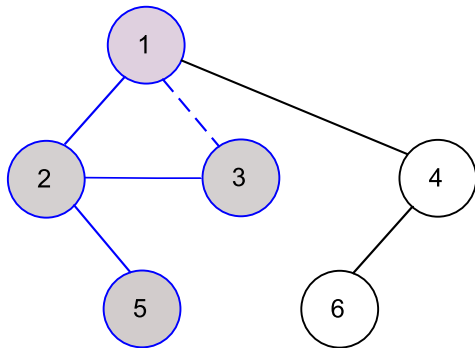
# Depth-First Search

Example



# Depth-First Search

Example

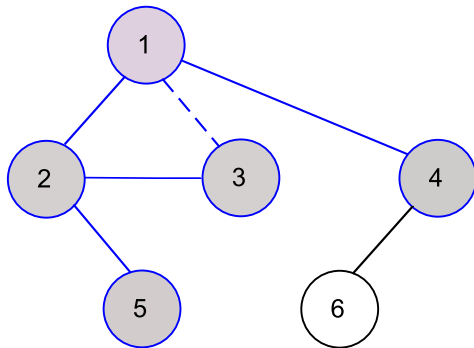


Stack



# Depth-First Search

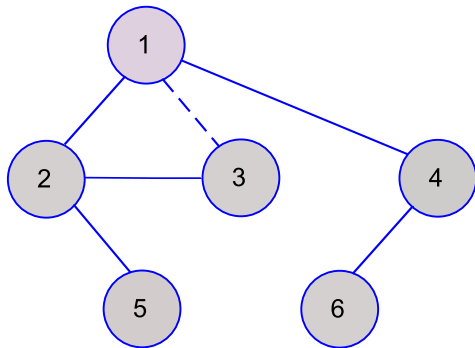
Example



Stack

# Depth-First Search

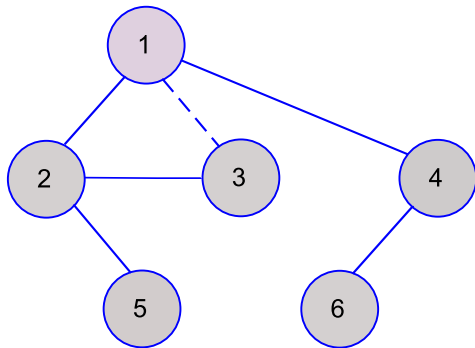
Example



Stack

# Depth-First Search

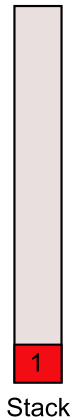
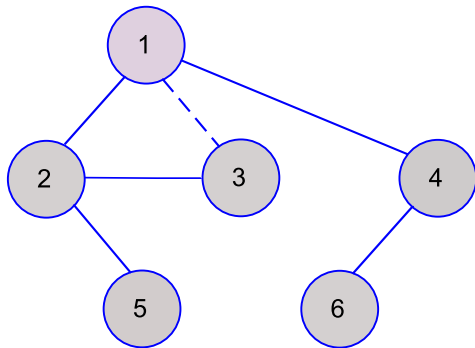
Example



Stack

# Depth-First Search

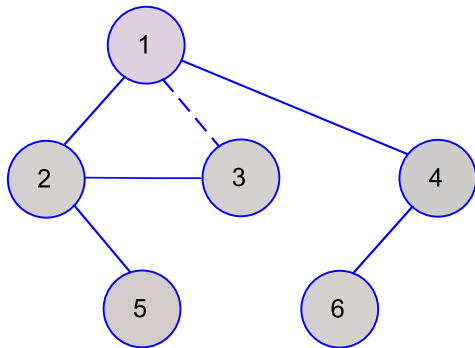
Example



Stack

# Depth-First Search

Example



Stack

# Depth-First Search

## Correctness

- Let  $G$  be the input graph, and suppose we start DFS from vertex  $s$  of  $G$ . Let  $v$  be some vertex of  $G$ .



# Depth-First Search

## Correctness

- Let  $G$  be the input graph, and suppose we start DFS from vertex  $s$  of  $G$ . Let  $v$  be some vertex of  $G$ .
- Let  $v$  be reachable from  $s$ . In our DFS, will  $v$  ever be visited?



# Depth-First Search

## Correctness

- Let  $G$  be the input graph, and suppose we start DFS from vertex  $s$  of  $G$ . Let  $v$  be some vertex of  $G$ .
- Let  $v$  be reachable from  $s$ . In our DFS, will  $v$  ever be visited?
- Let  $v$  be **not** reachable from  $s$ . In our DFS, will  $v$  ever be visited?



# Depth-First Search

## Complexity

### The Algorithm

- 1 Mark  $s$  as visited. Push  $s$  on to the stack.
  - 2 Repeat the following till the stack is empty:
    - Pop a vertex (say  $w$ ) from the stack.
    - For each neighbour  $x$  of  $w$  that is not yet visited,
      - Mark  $x$  as visited, and push  $x$  on to the stack.
- For a graph with  $n$  vertices and  $m$  edges, how much **space** does this algorithm use?



# Depth-First Search

## Complexity

### The Algorithm

- 1 Mark  $s$  as visited. Push  $s$  on to the stack.
  - 2 Repeat the following till the stack is empty:
    - Pop a vertex (say  $w$ ) from the stack.
    - For each neighbour  $x$  of  $w$  that is not yet visited,
      - Mark  $x$  as visited, and push  $x$  on to the stack.
- For a graph with  $n$  vertices and  $m$  edges, how much **time** does this algorithm take?



# Depth-First Search

## Complexity

Graph with $n$ vertices, $m$ edges	Adjacency Matrix	Adjacency List
Space required	$O(n^2)$	$O(n + m)$
Time required	$O(n^2)$	$O(n + m)$

- Adjacency list – advantageous especially for *sparse* graphs, where  $m \ll n^2$ .

