

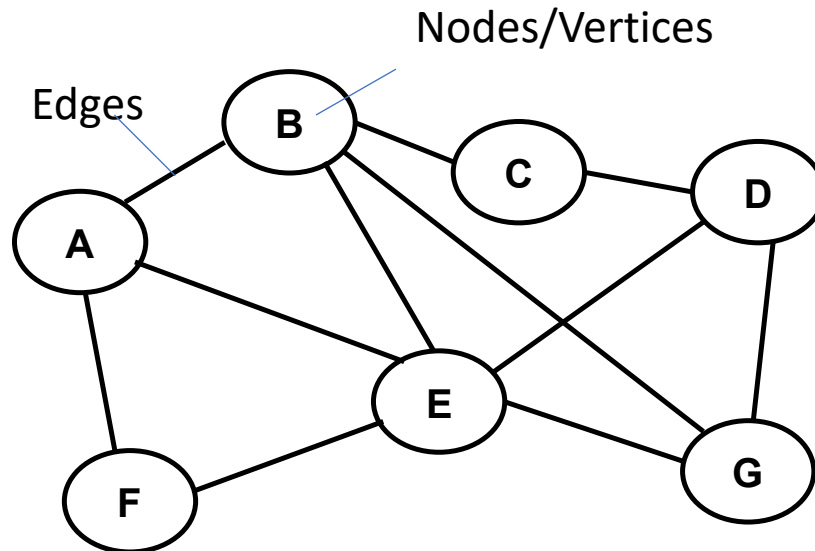
Graph Algorithms: Breadth First Search (BFS)

Instructor: Krishna Venkatasubramanian

CSC 212

Graphs

- Data structures that connect a set of objects to form a kind of a network
- Objects are called **“Nodes”** or **“Vertices”**
- Connections are called **“Edges”**
- **Unlike trees graphs may have paths that form loops like “A->B->C-> A”**



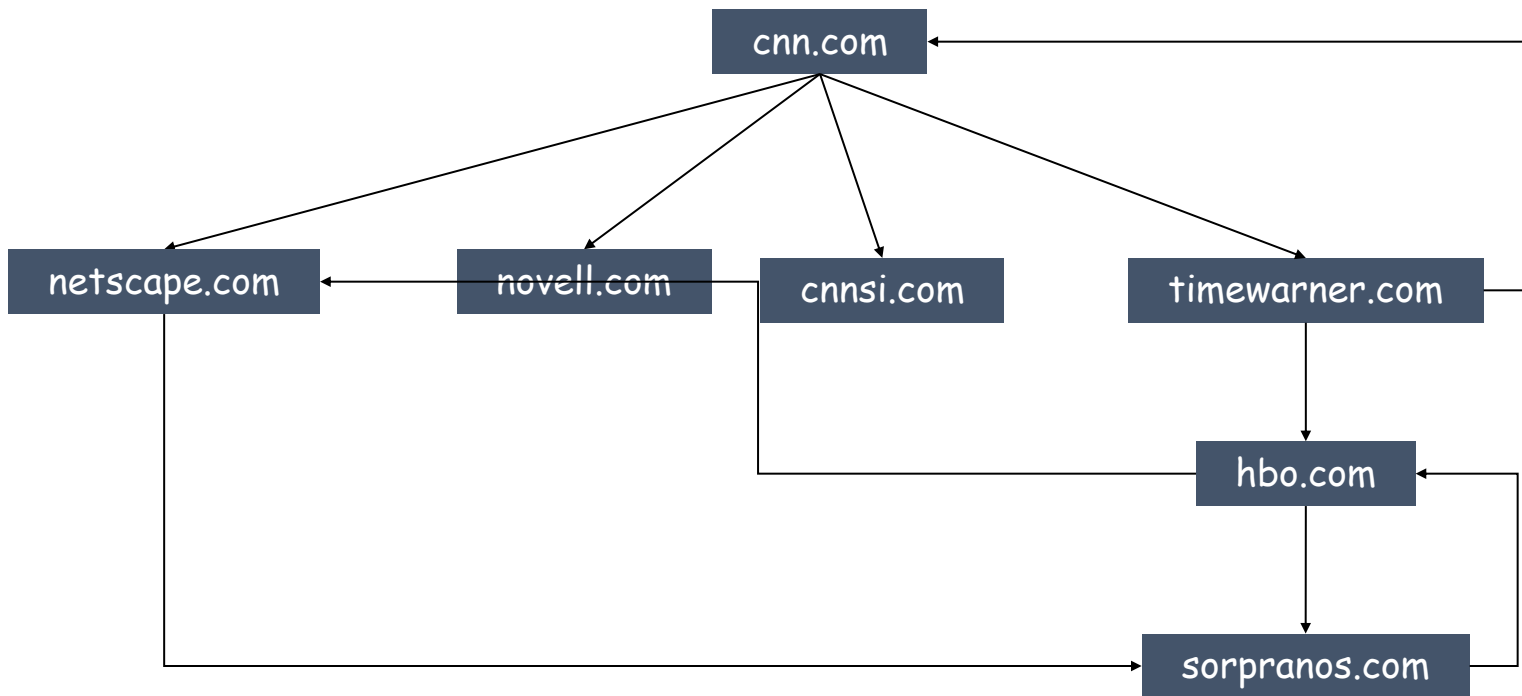
Some Graph Applications

<i>Graph</i>	<i>Nodes</i>	<i>Edges</i>
transportation	street intersections	highways
communication	computers	fiber optic cables
World Wide Web	web pages	hyperlinks
social	people	relationships
food web	species	predator-prey
software systems	functions	function calls
scheduling	tasks	precedence constraints
circuits	gates	wires

World Wide Web

Web graph.

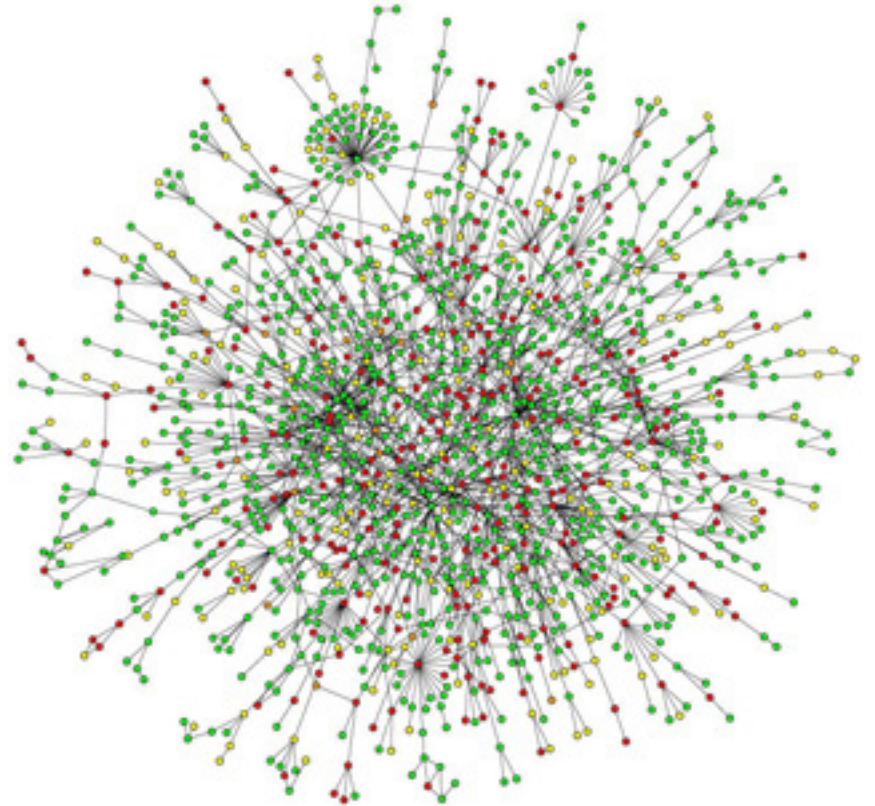
- Node: web page.
- Edge: hyperlink from one page to another.



Protein Networks

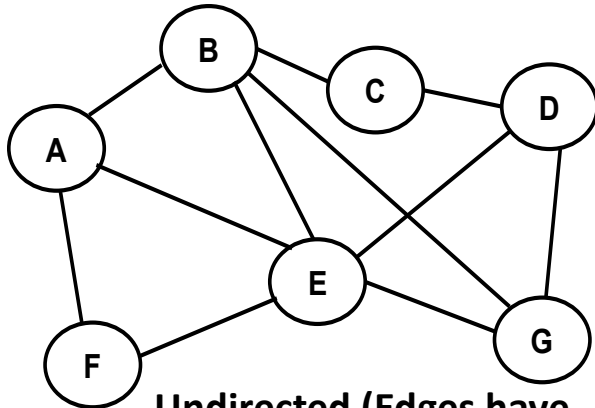
Nodes are proteins

**Edges are connections
(interaction between
proteins)**

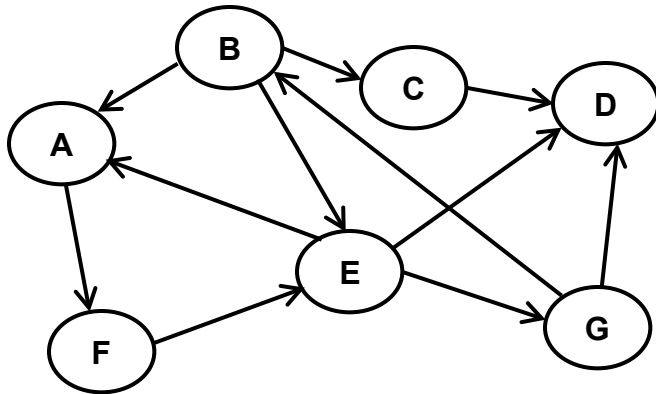


Types of Graphs

Directed vs. undirected

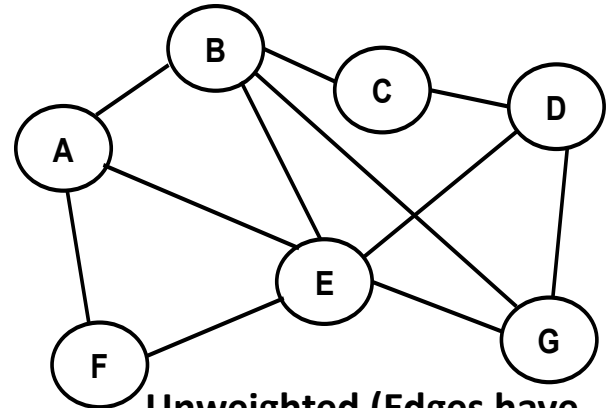


Undirected (Edges have no direction)

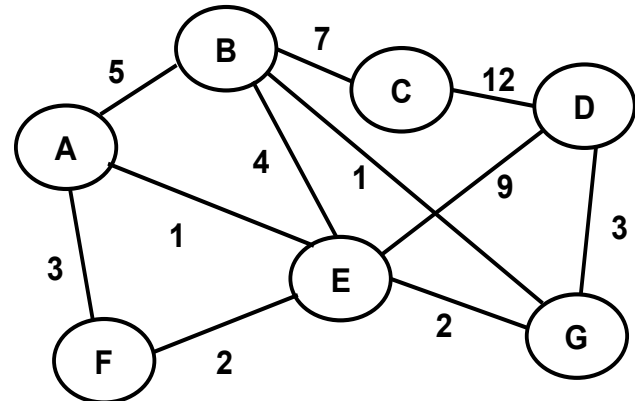


Directed (Edges have directions)

Weighted vs. unweighted



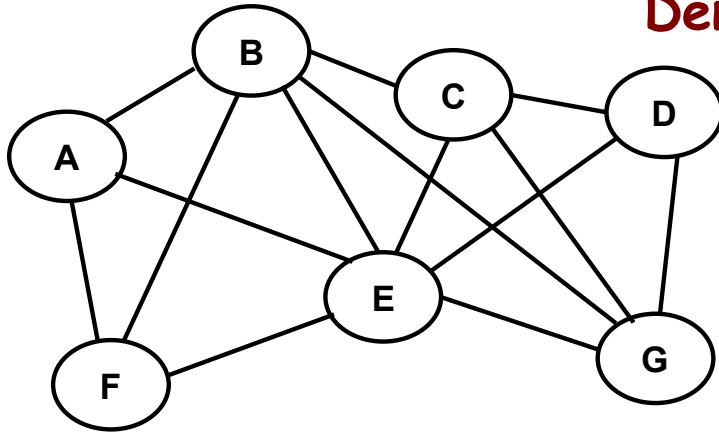
Unweighted (Edges have no cost/weight)



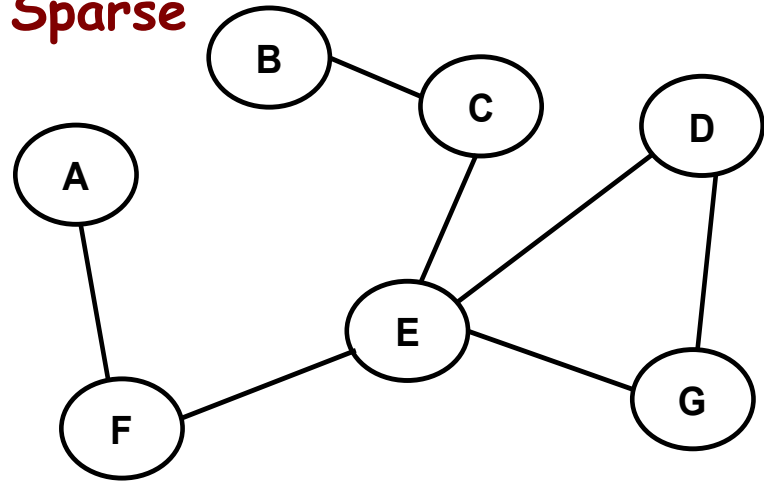
Weighted (Edges have associated cost/weight)

Types of Graphs (Cont'd)

Dense vs. Sparse



Dense graphs (many edges between nodes)



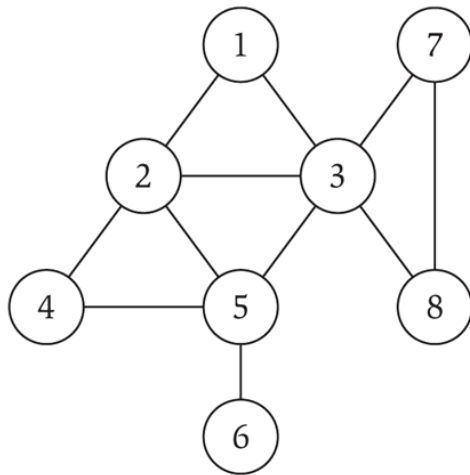
Sparse graphs (few edges between nodes)

- If the graph has **n vertices** (nodes) \Rightarrow Maximum # of edges is $(n^2-n)/2 = O(n^2)$
- In dense graphs number of edges is close to $O(n^2)$
- In sparse graphs number of edges is close to $O(n)$

Undirected Graphs

Undirected graph. $G = (V, E)$

- V = set of nodes or vertices
- E = edges between pairs of nodes.
- Graph size parameters: $n = |V|$, $m = |E|$.



$V = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$

$E = \{ 1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6 \}$

$n = |V| = 8$

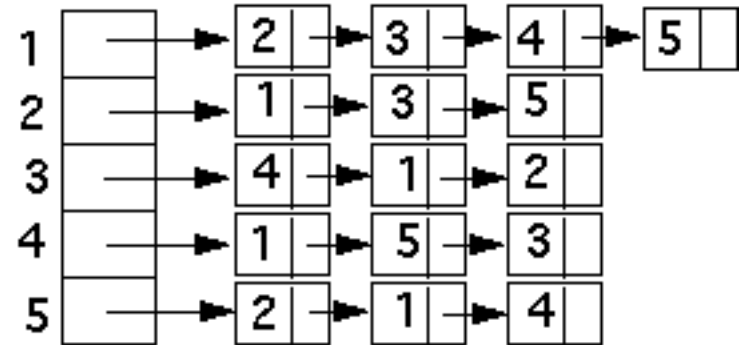
$m = |E| = 11$

Graph Representation

Two main methods

	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	0	0	1	1
C	1	0	0	0	0	1
D	1	0	0	0	0	0
E	0	1	0	0	0	0
F	0	1	1	0	0	0

Adjacency Matrix

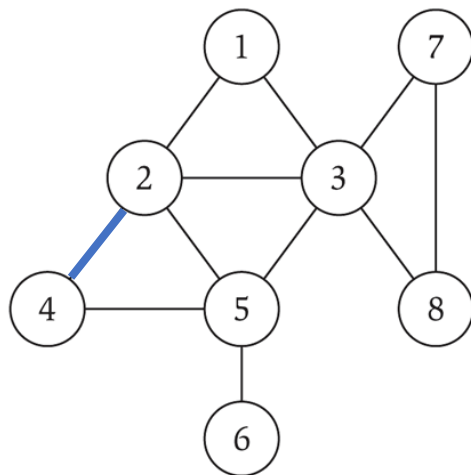


Adjacency List

Adjacency Matrix

Adjacency matrix. $|V|$ -by- $|V|$ matrix (A)

- $A[i, j] = 1$ if exists edge between node i and node j
- Space proportional to $|V|^2$
- Checking if (u, v) is an edge takes $O(1)$ time.
- Identifying all edges takes $O(|V|^2)$ time.
- For undirected graph \rightarrow matrix is symmetric across the diagonal



Vertices

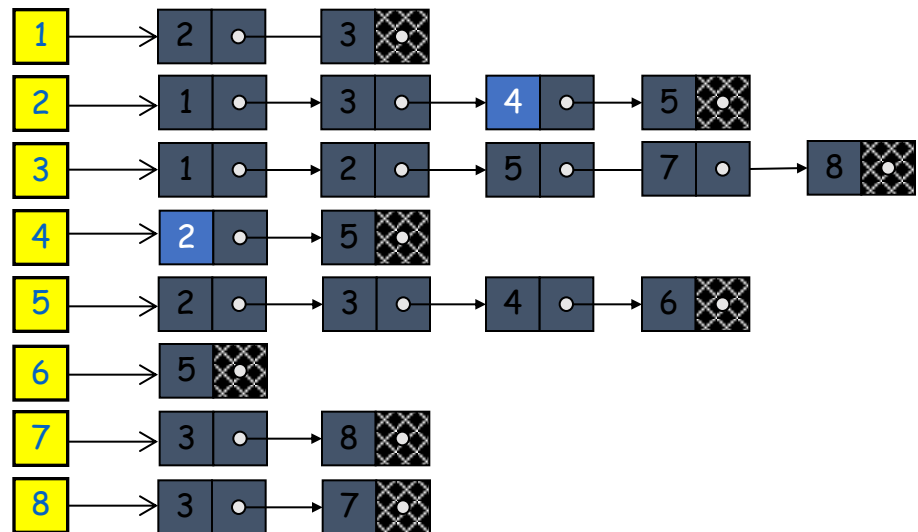
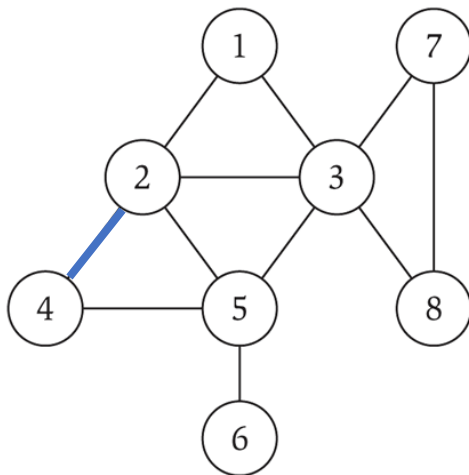
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Vertices

Adjacency List

Adjacency list. Node indexed array of lists.

- Two representations of each edge.
- Space proportional to $O(E + V)$.
- Checking if (u, v) is an edge takes $O(\deg(u))$ time. degree = number of neighbors of u
- Identifying all edges takes $O(E + V)$ time.

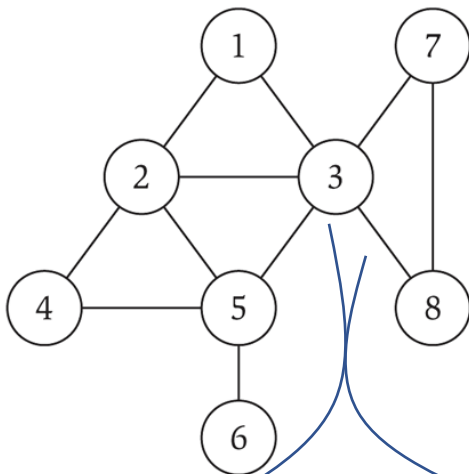


Degree of a Node

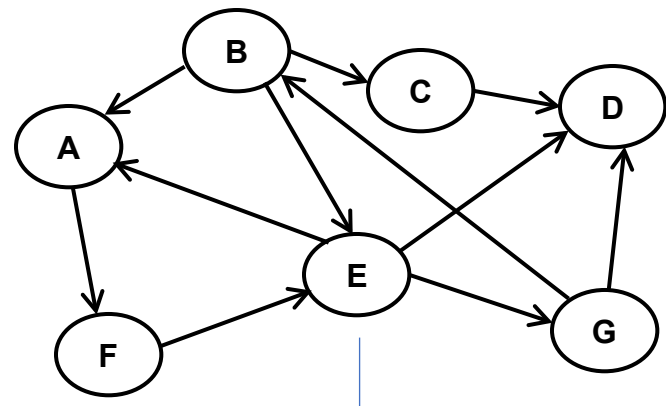
In-degree(v): Number of edges coming to (entering) node v

Out-degree(v): Number of edges getting out (leaving) node v

For Undirected graphs \rightarrow In-degree = Out-degree



In-degree(3) = Out-degree(3) = 5

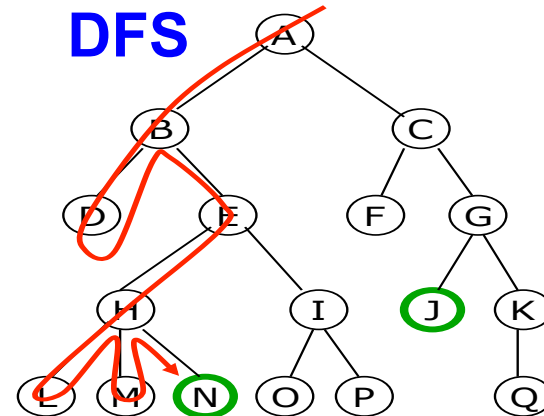
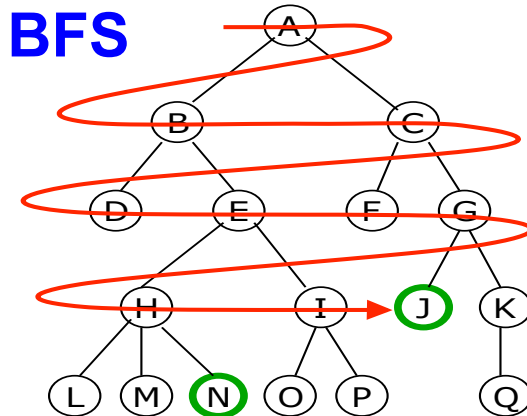


In-degree(E) = 2
Out-degree(E) = 3

Each vertex will have different In-Degree and Out-Degree

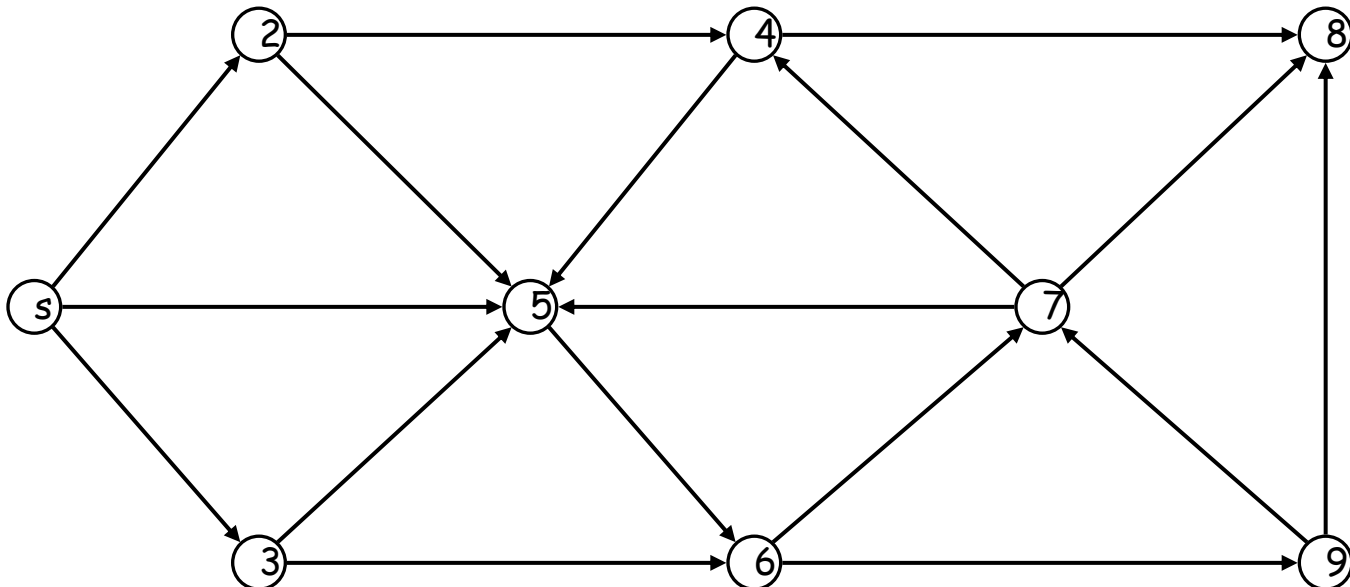
Graph Traversal

- ◆ Graph Traversal means visiting each node in the graph
- ◆ There is a starting node (s)
- ◆ Two main types of traversal
 - ◆ Breadth-First-Search (BFS)
 - ◆ Depth-First-Search (DFS)
- ◆ Both are applicable for *directed* and *undirected graphs*



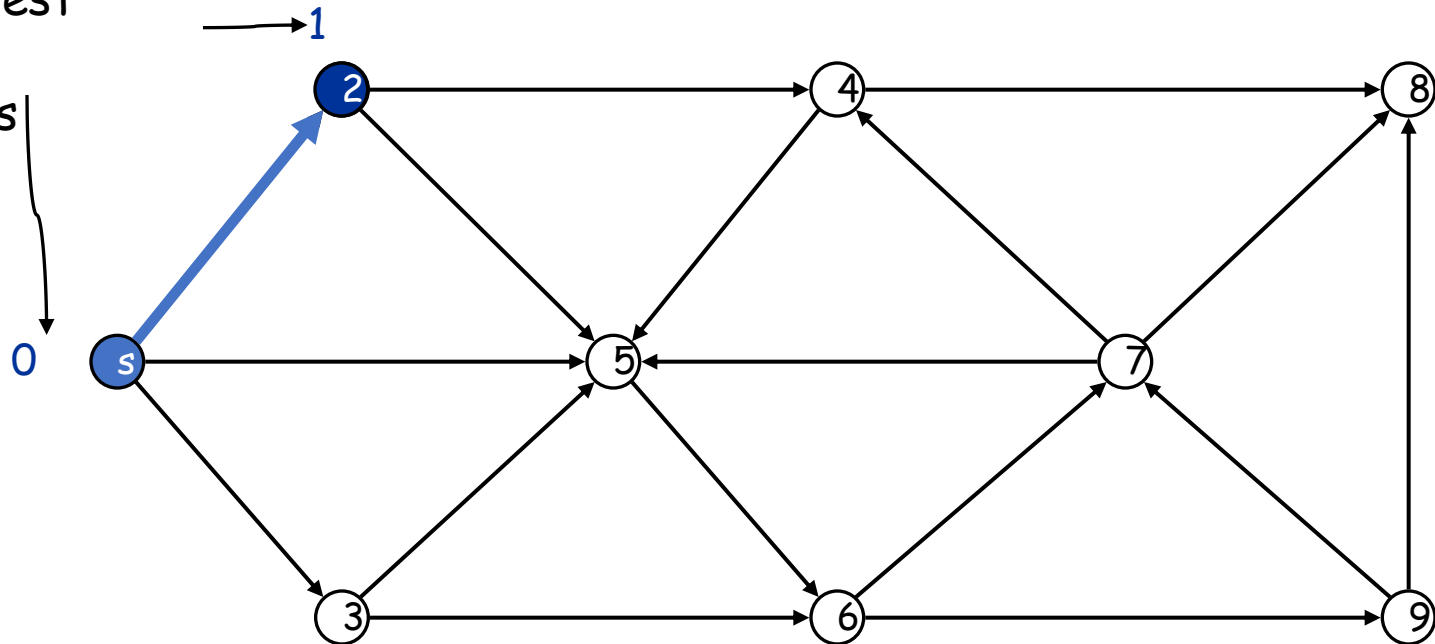
BFS i.e., Breadth-First Traversal

- Even though the approach is called **Breadth-First Search (BFS)** it is essentially a traversal algorithm.
 - The idea is to traverse the graph in a specific way, which can be leveraged for search
 - As we are traversing the graph we can check if the node we are on has what we want
- Visit the nodes one-level at a time
- Requires a **queue** (First-come-first-served)



BFS Example

Shortest
path
from s



Undiscovered

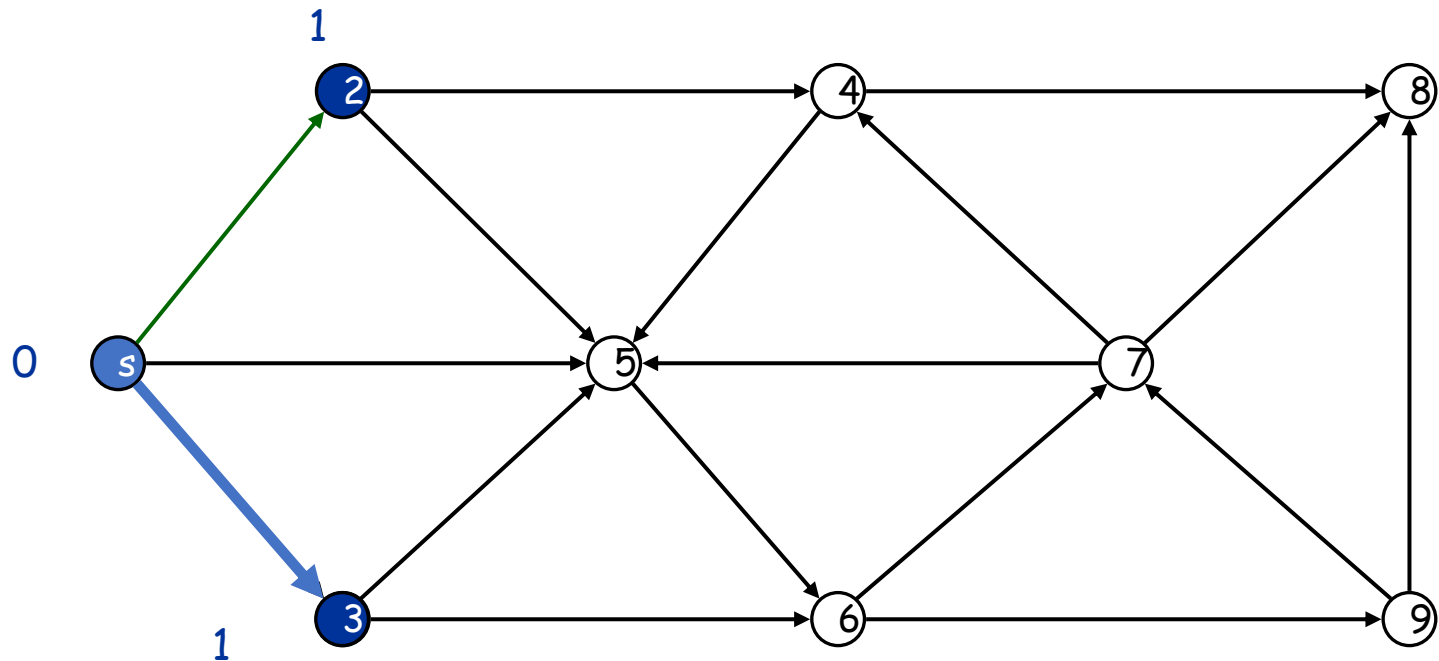
Discovered

Top of queue

Finished

Queue: s

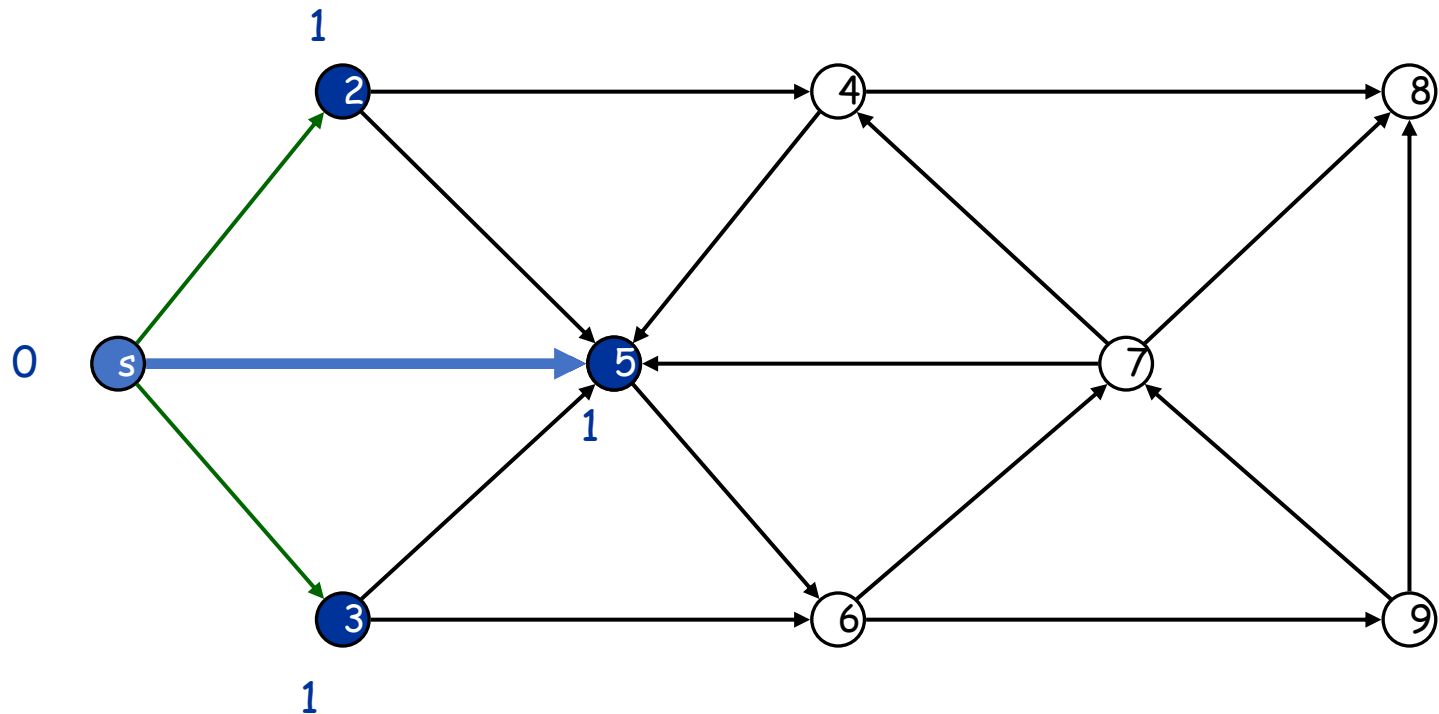
BSF Example



Undiscovered
Discovered
Top of queue
Finished

Queue: s 2

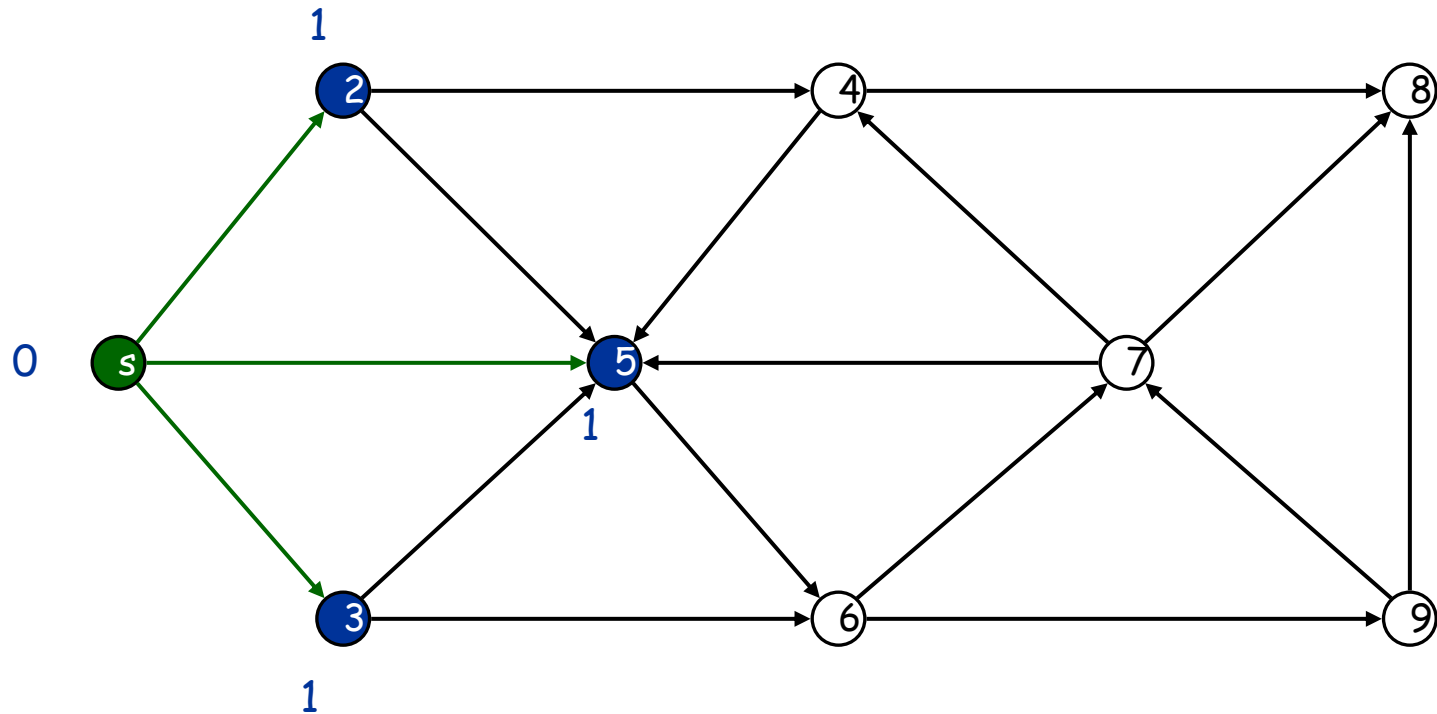
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: s 2 3

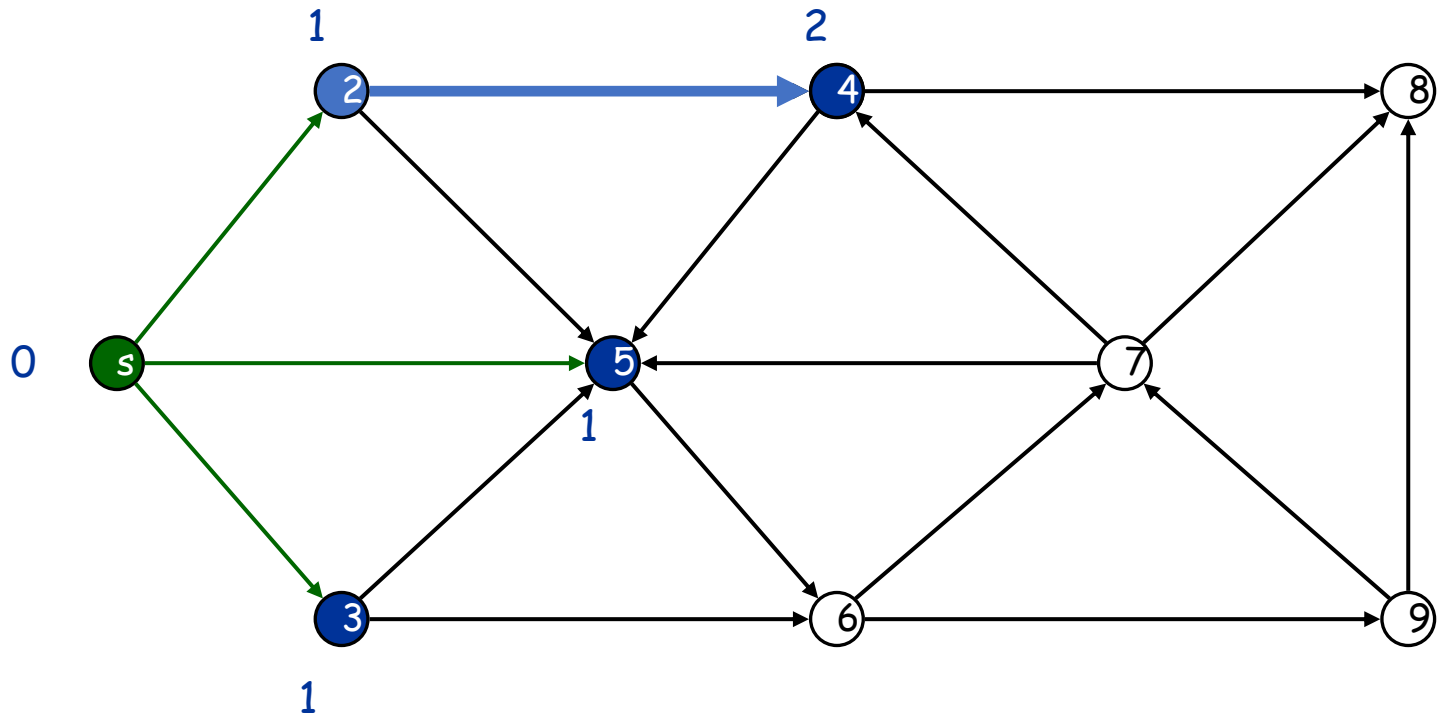
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 2 3 5

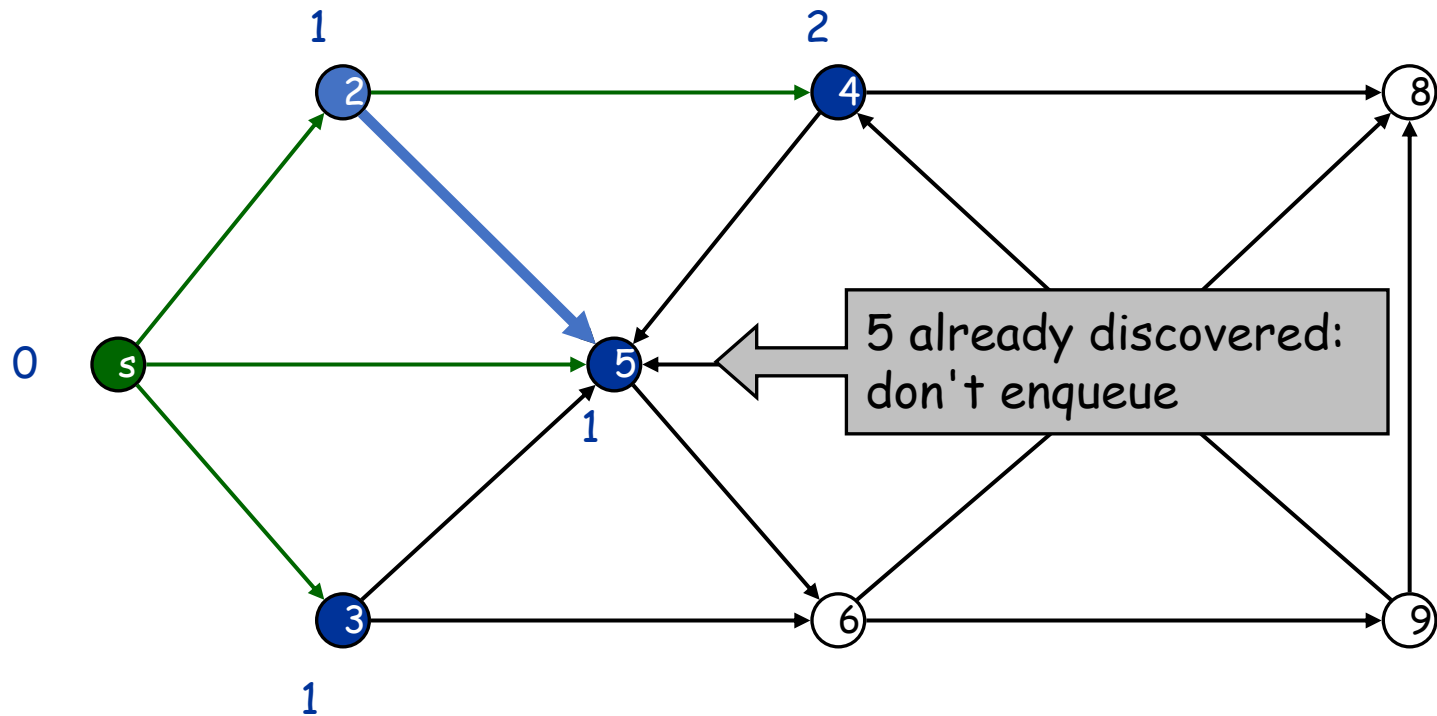
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 2 3 5

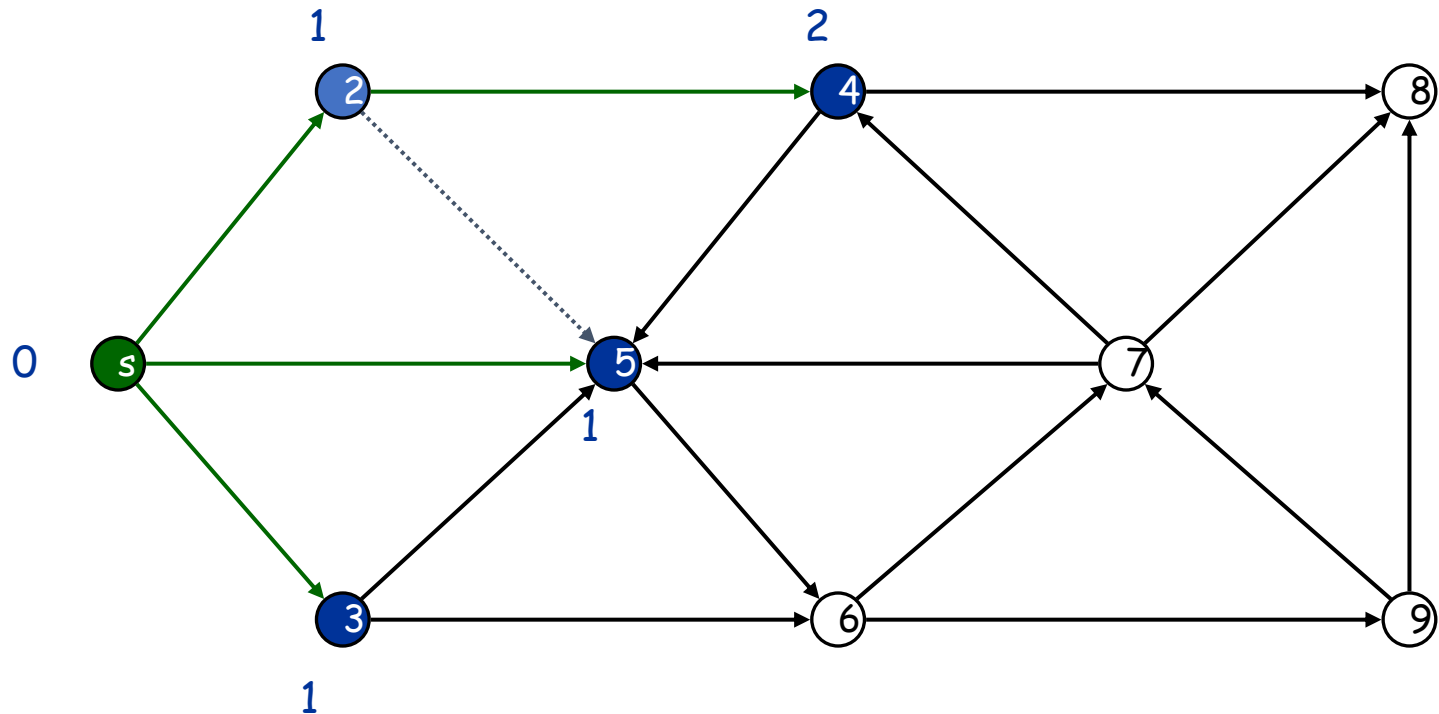
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 2 3 5 4

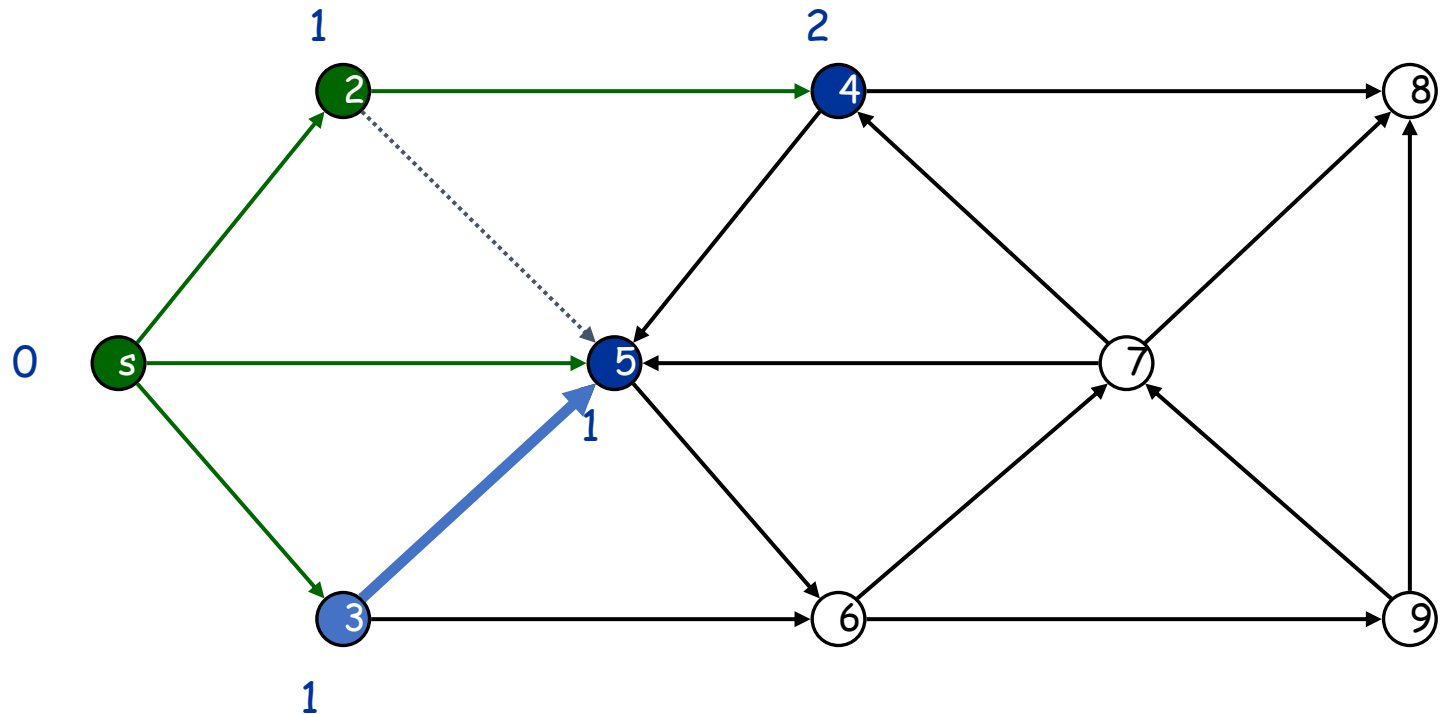
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 2 3 5 4

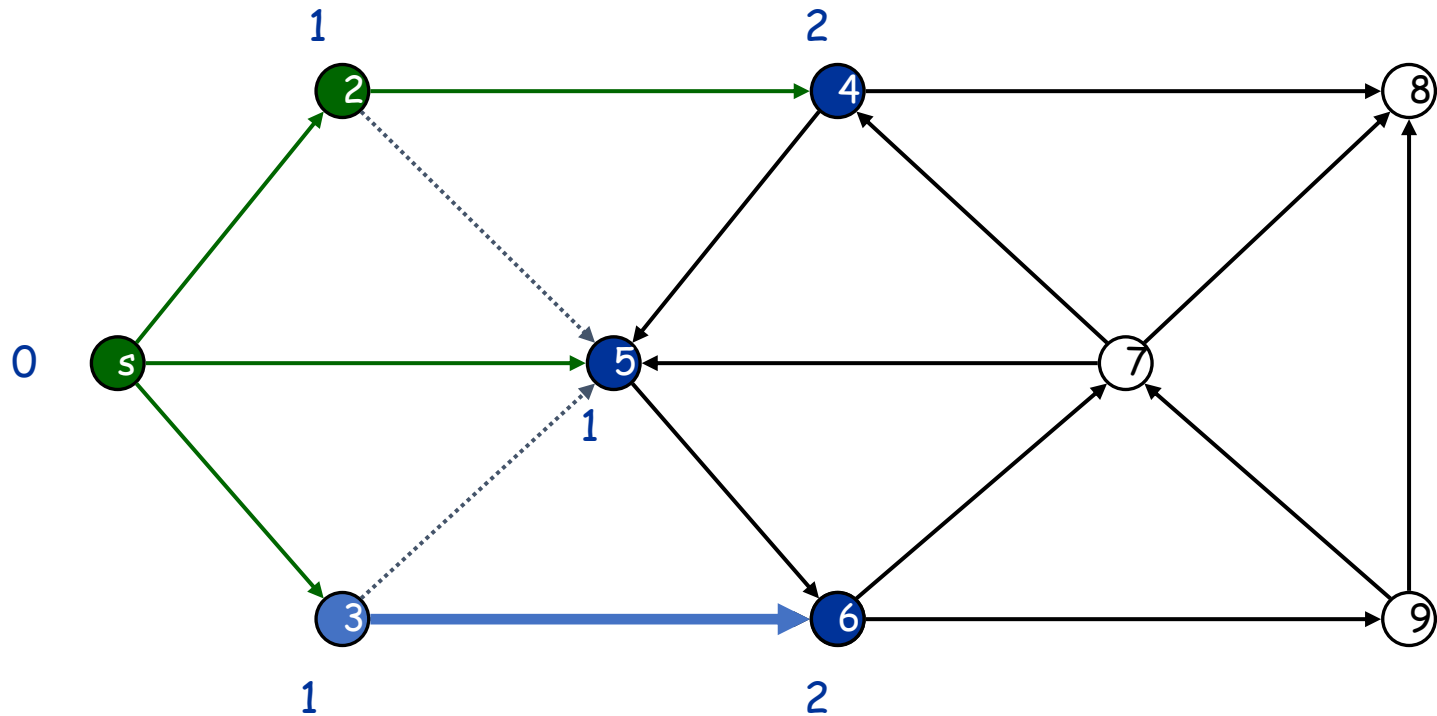
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 3 5 4

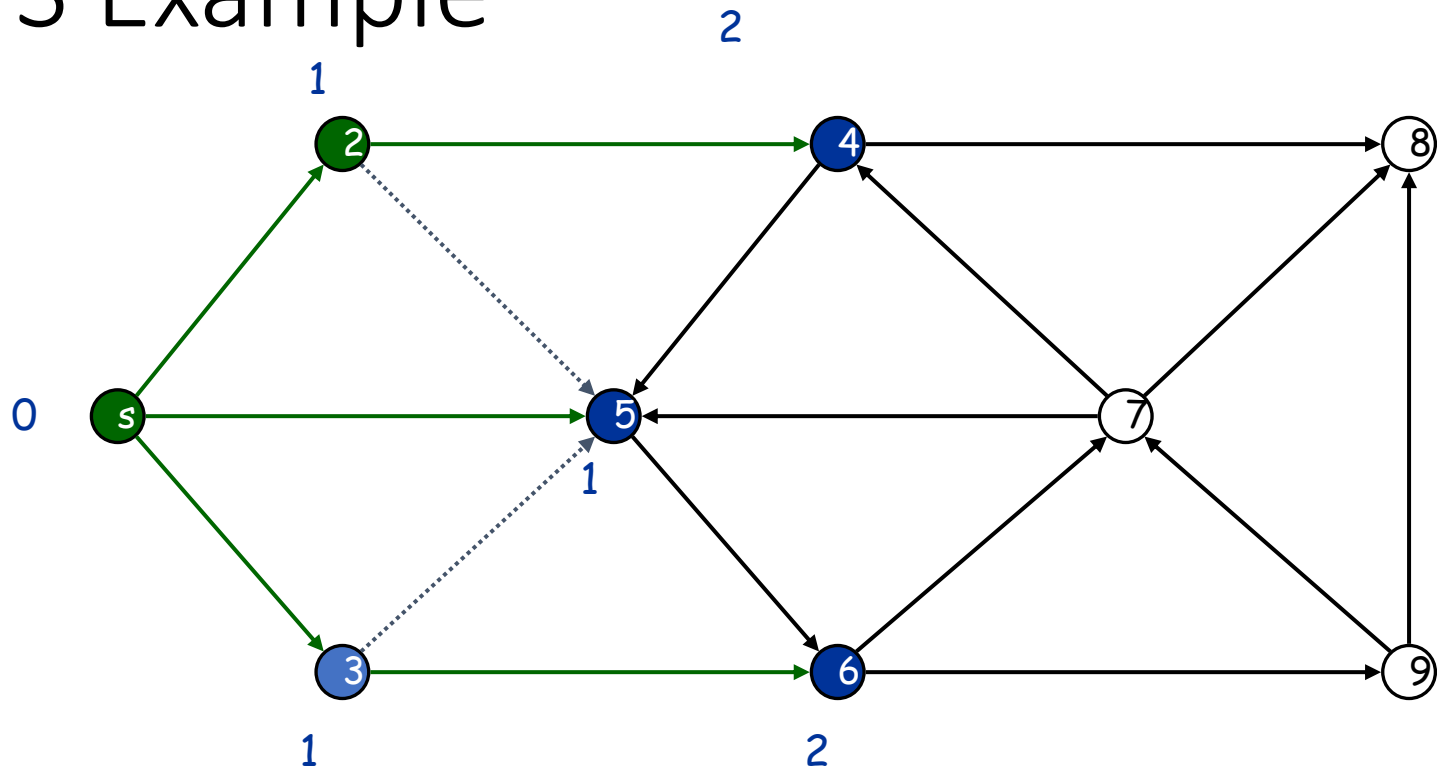
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 3 5 4

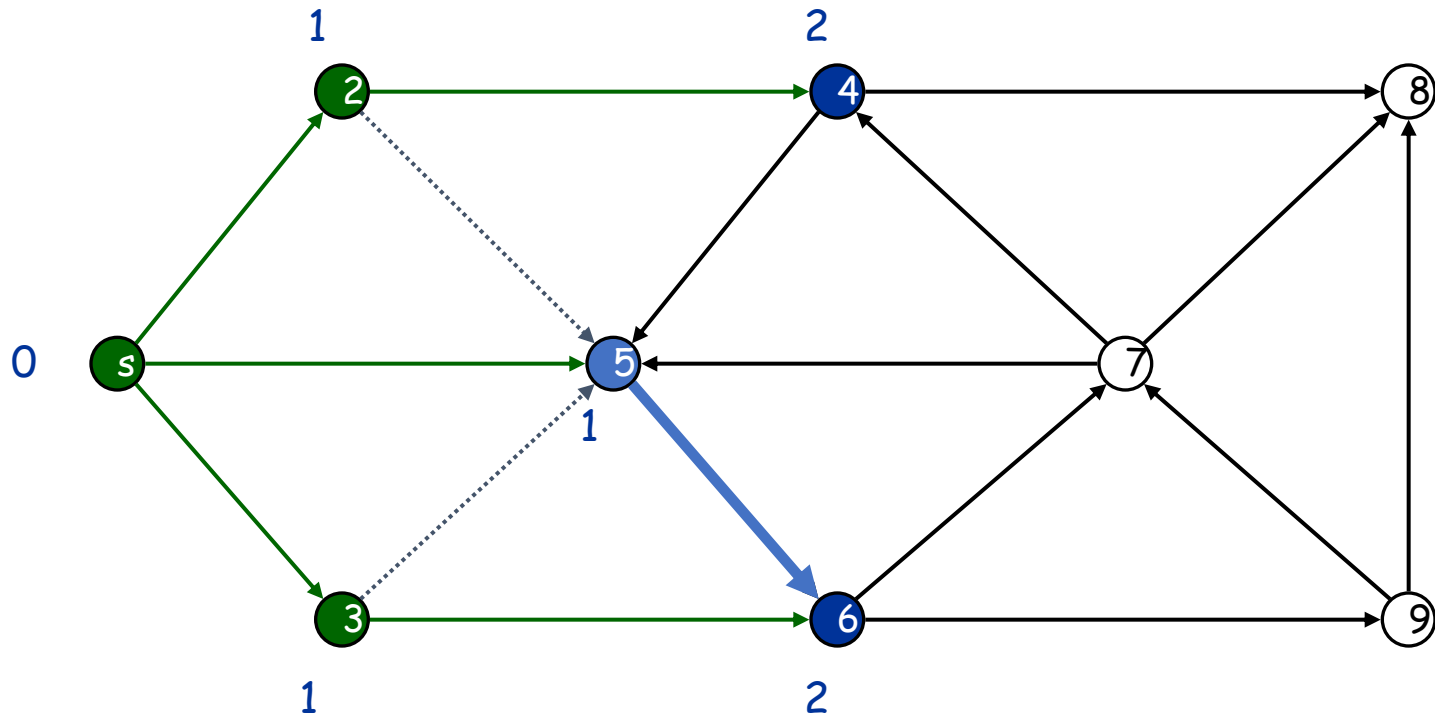
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 3 5 4 6

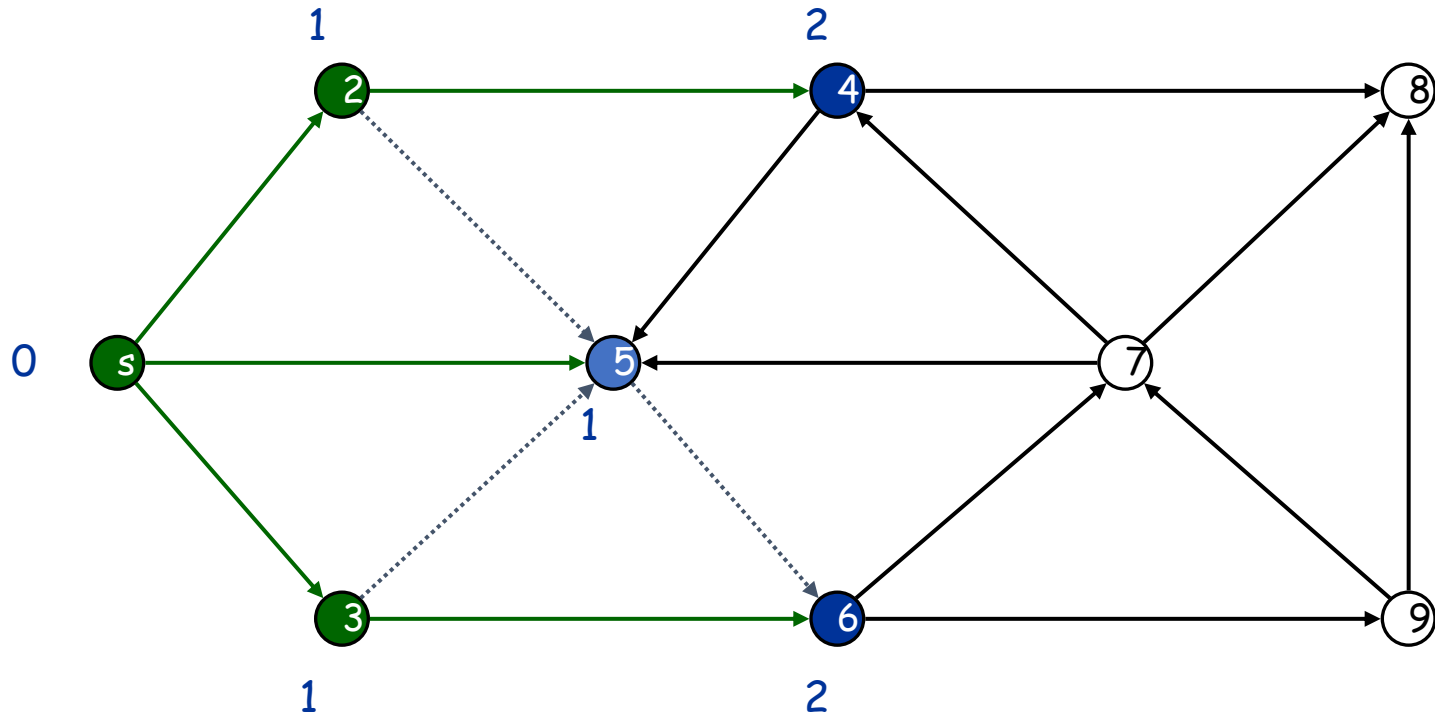
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 5 4 6

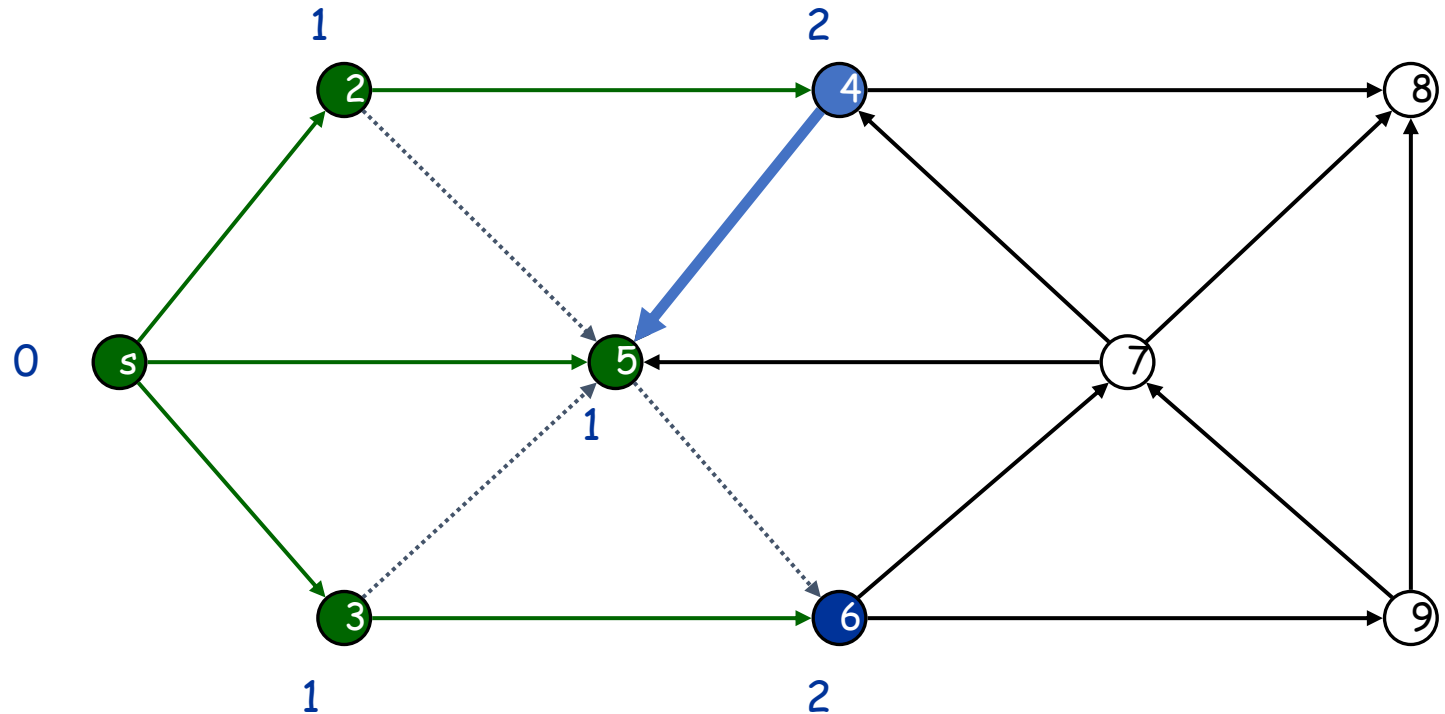
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 5 4 6

BFS Example



Undiscovered

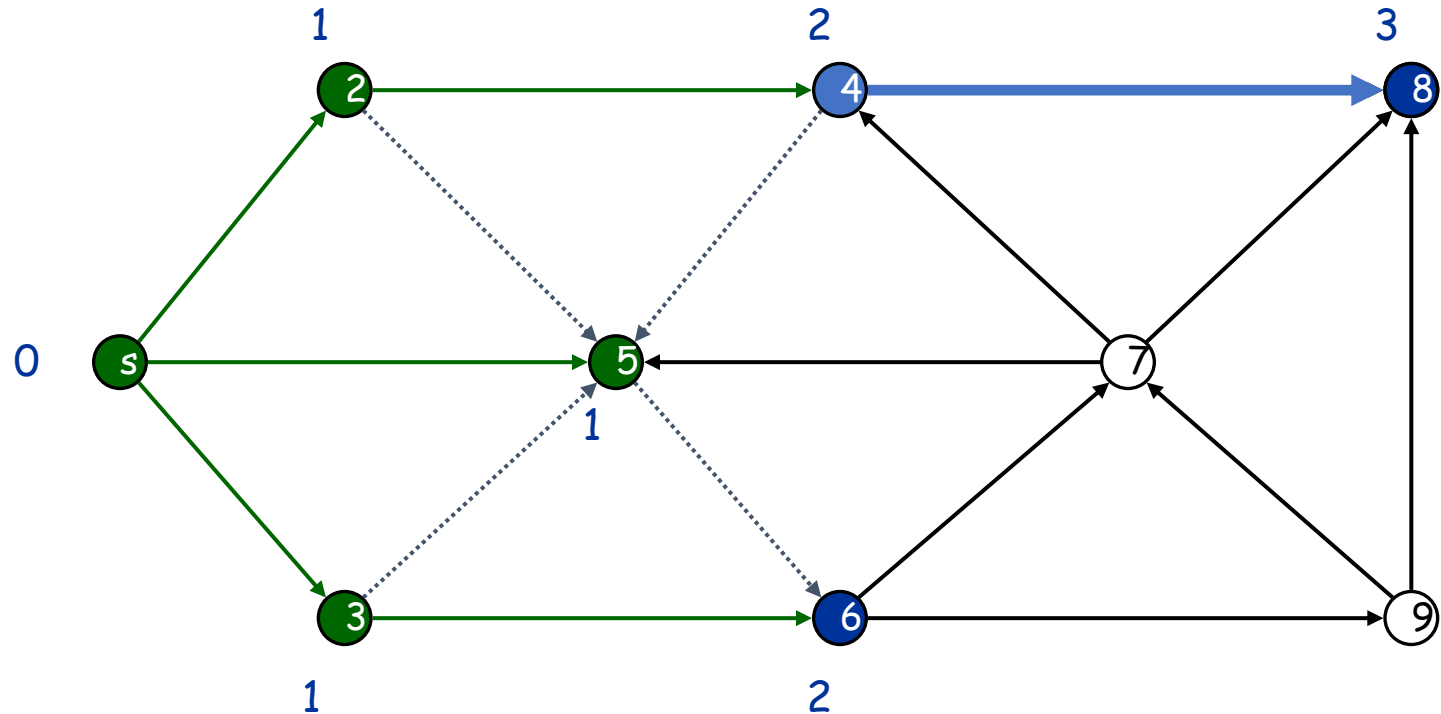
Discovered

Top of queue

Finished

Queue: 4 6

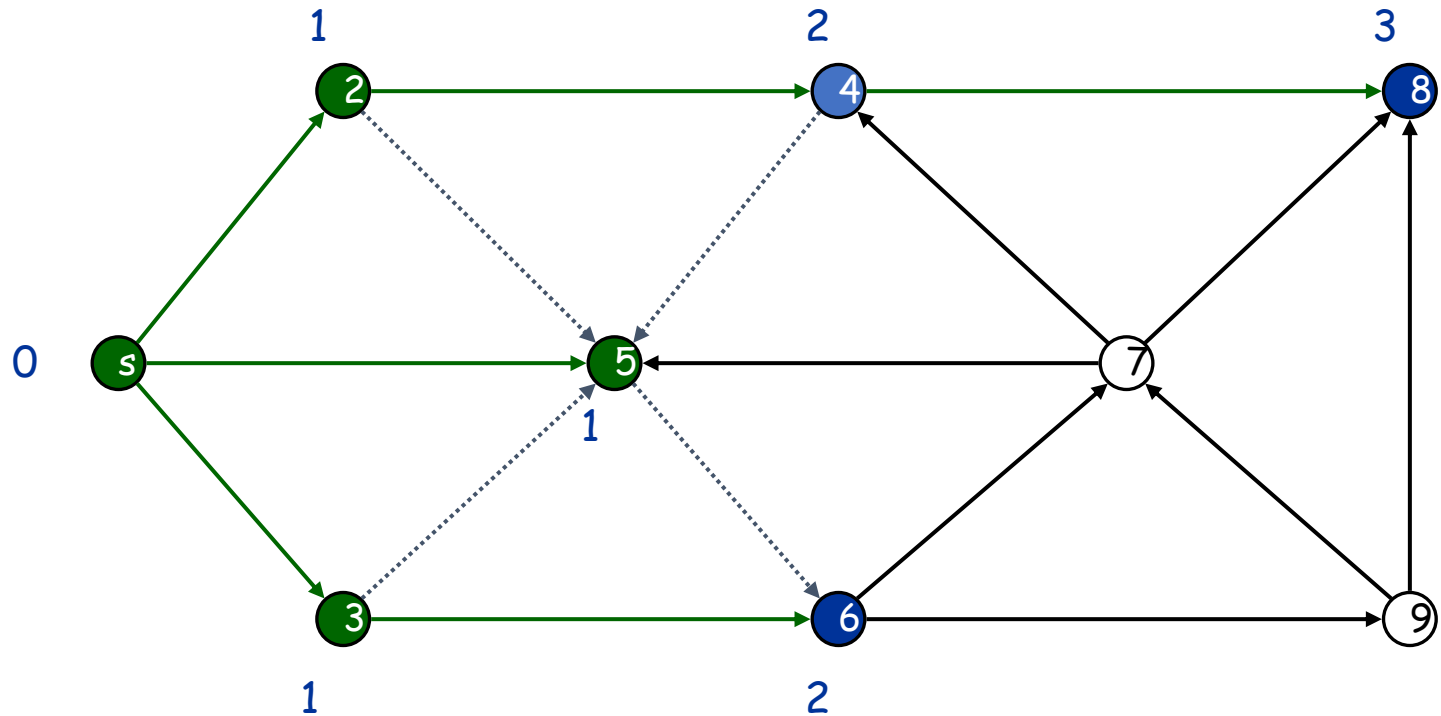
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 4 6

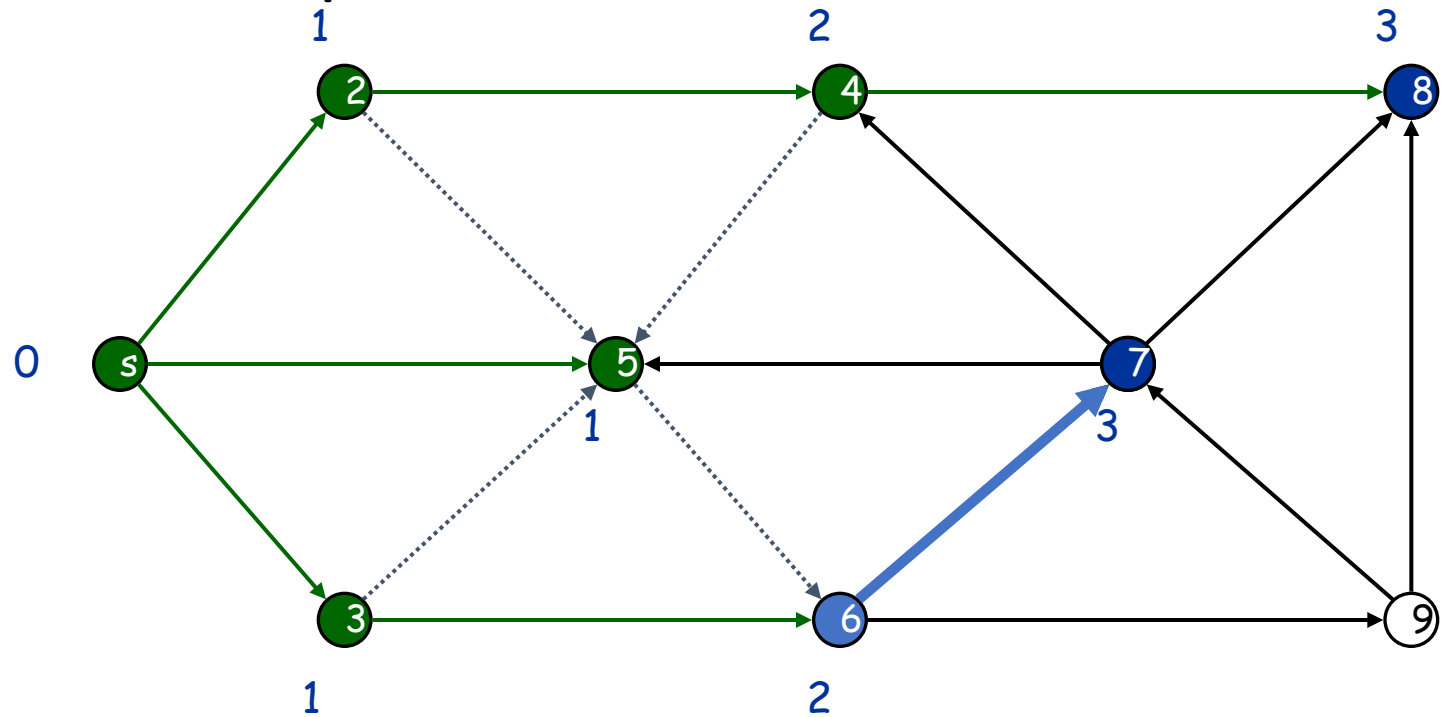
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 4 6 8

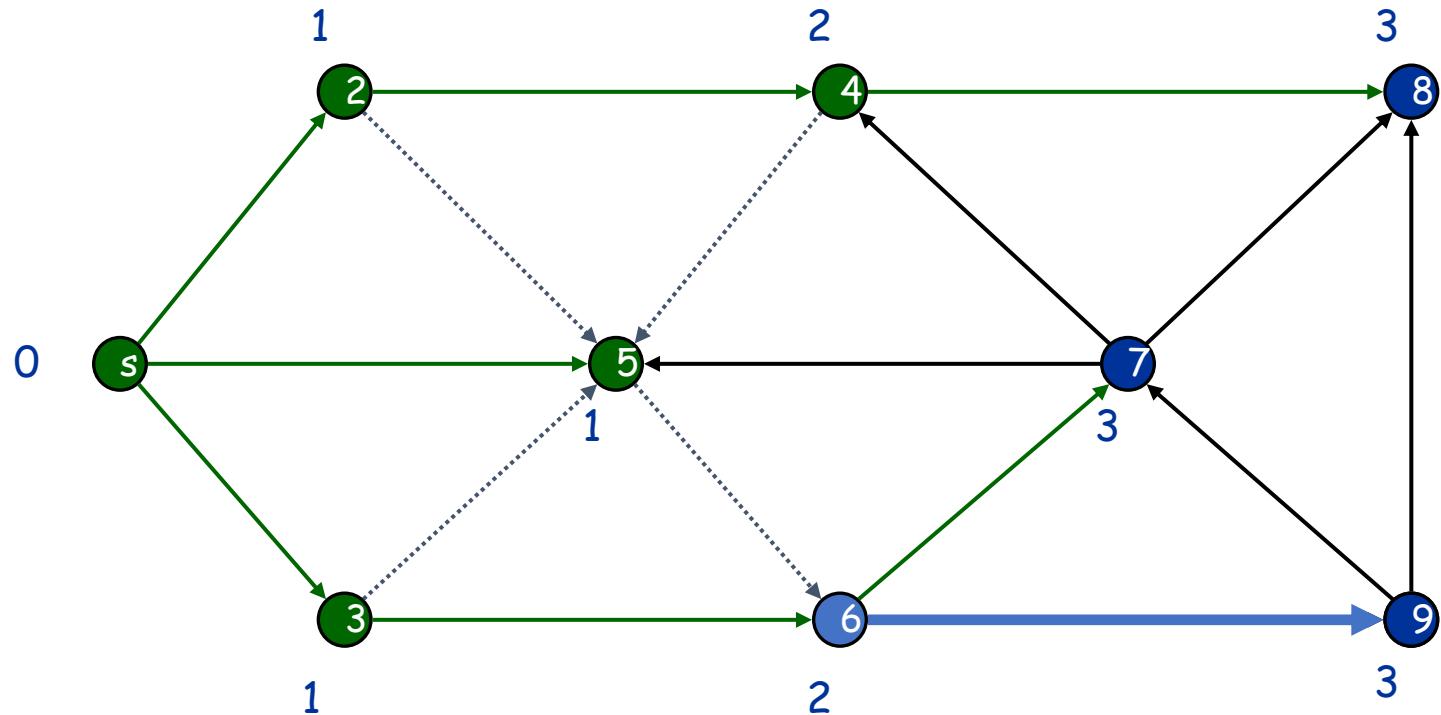
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 6 8

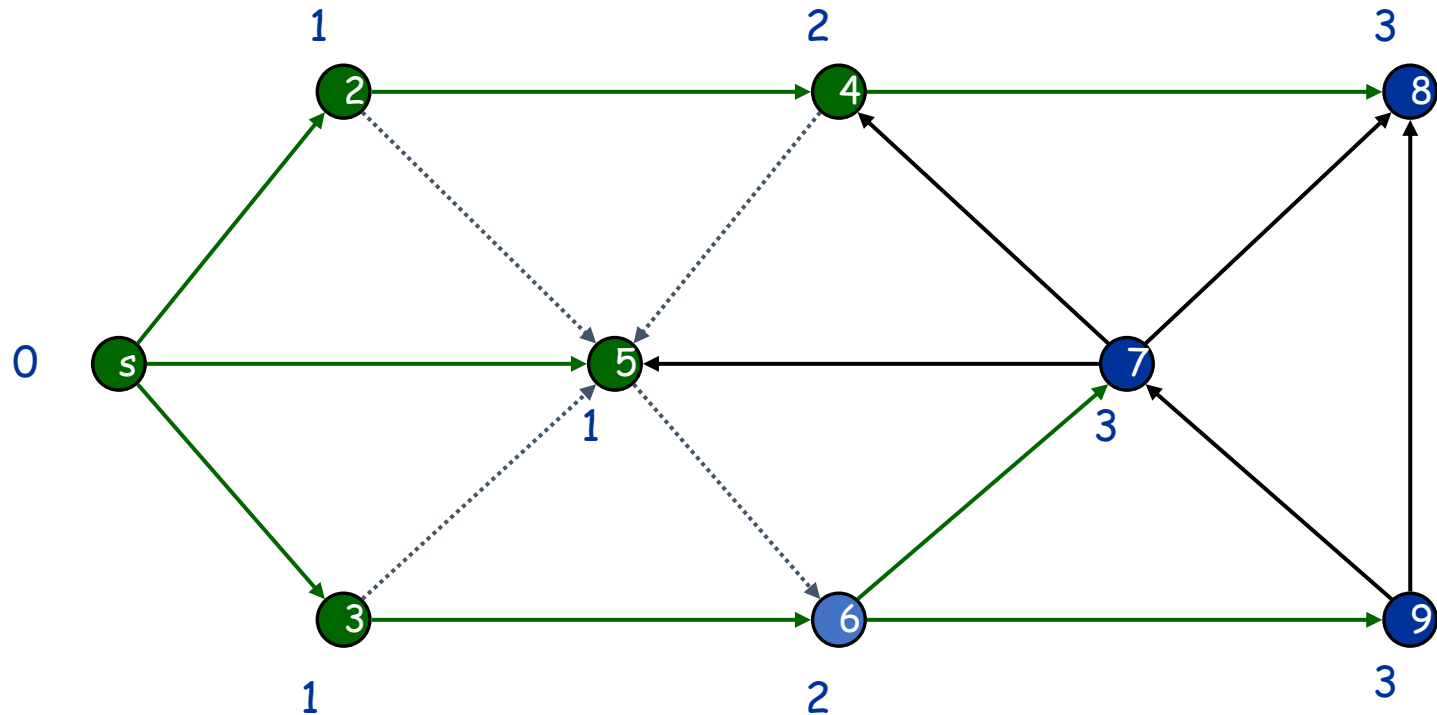
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 6 8 7

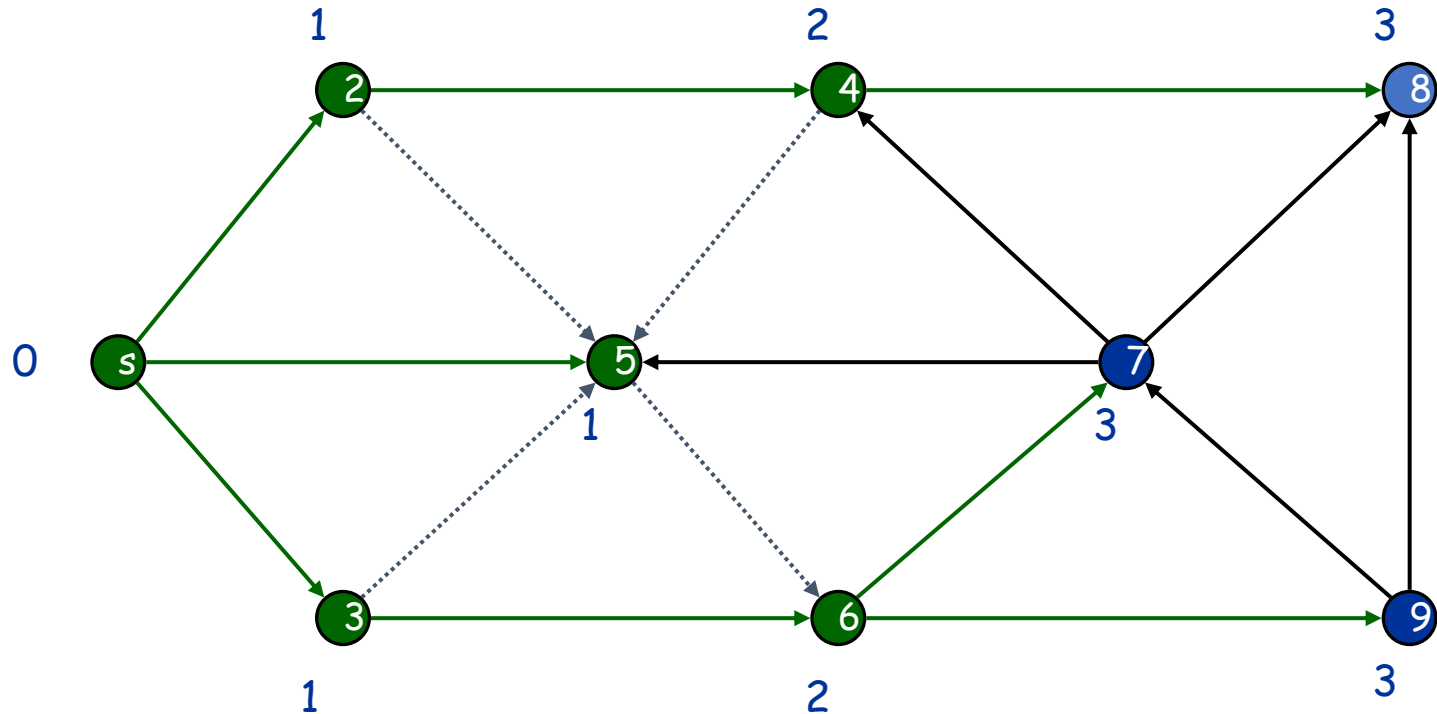
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 6 8 7 9

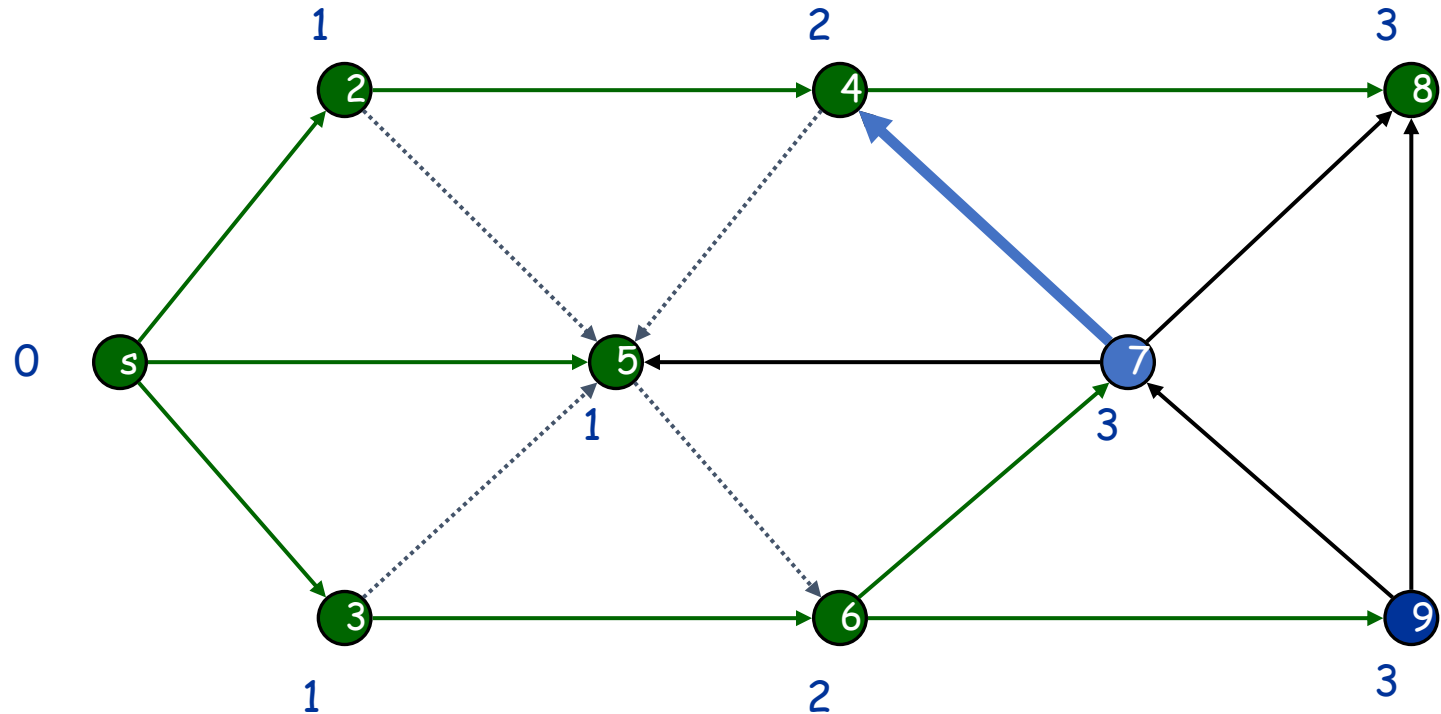
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 8 7 9

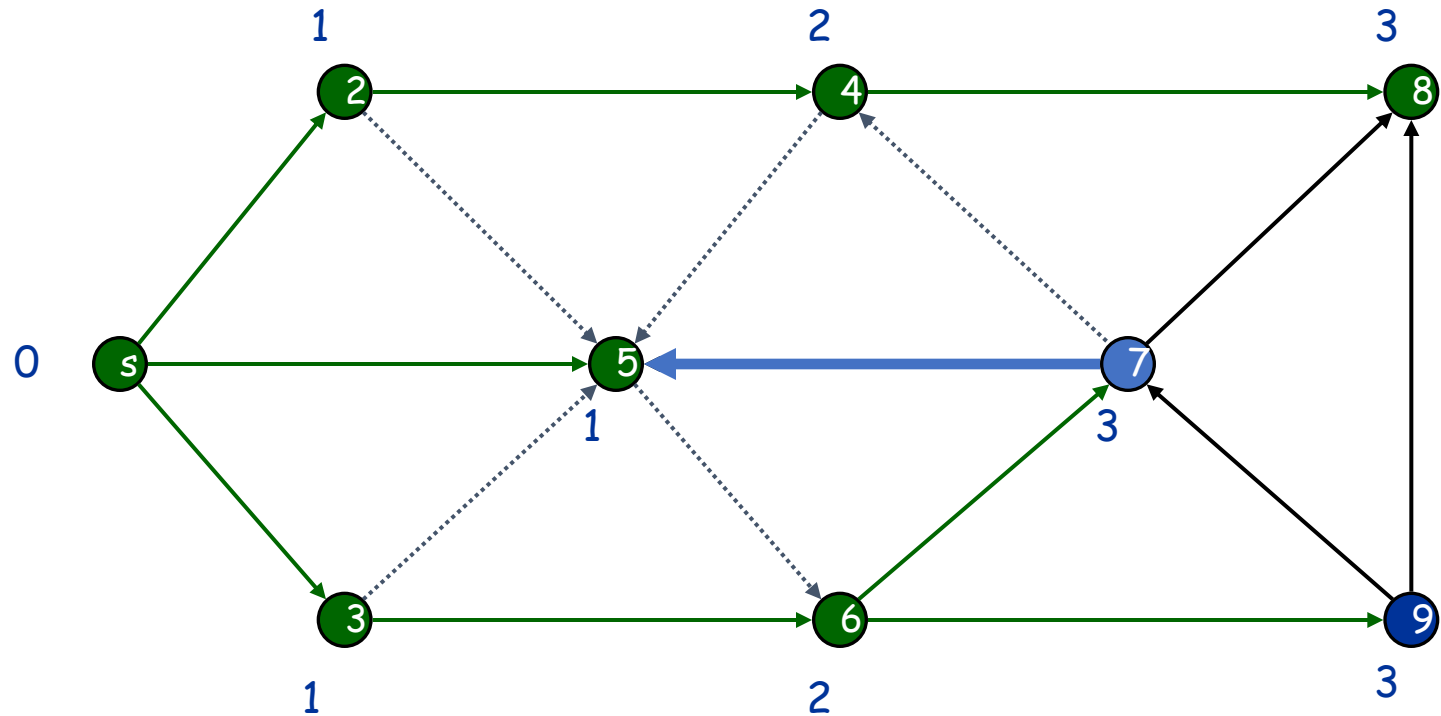
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 7 9

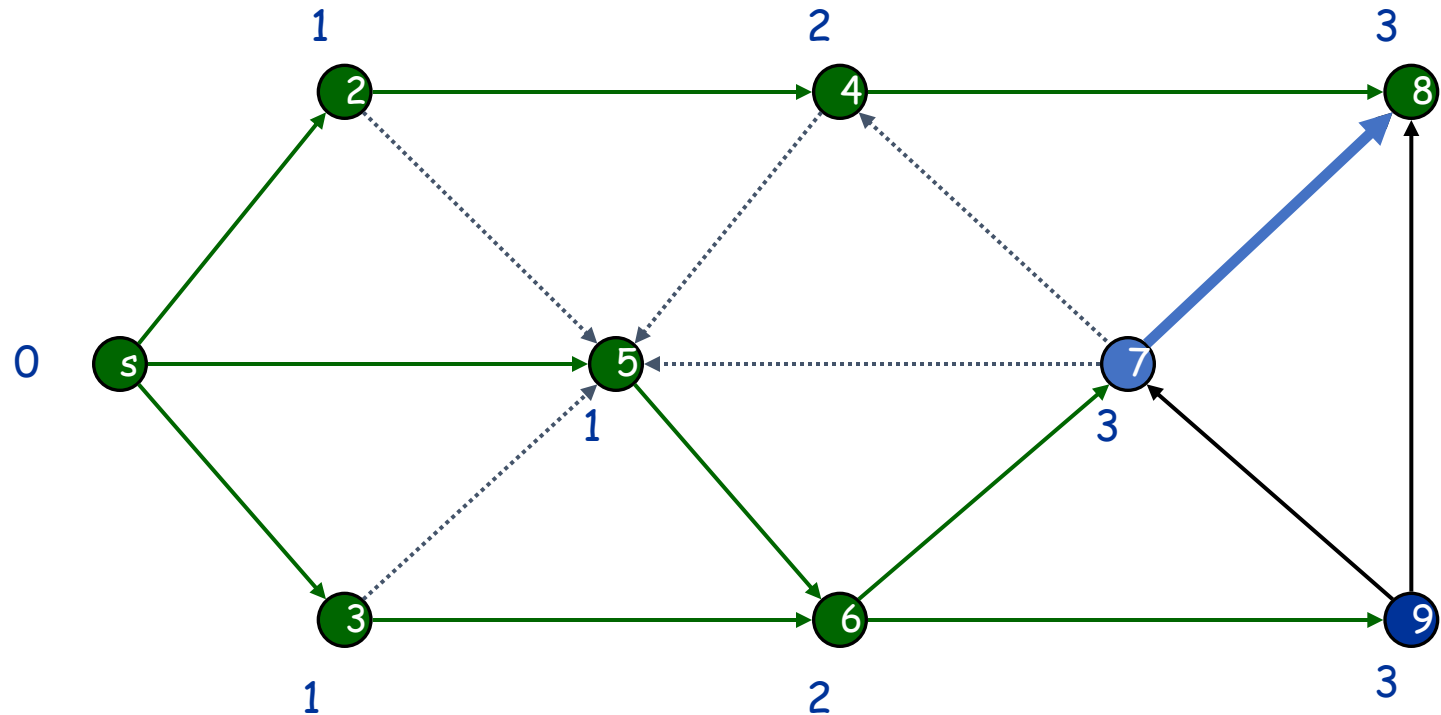
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 7 9

BFS Example



Undiscovered

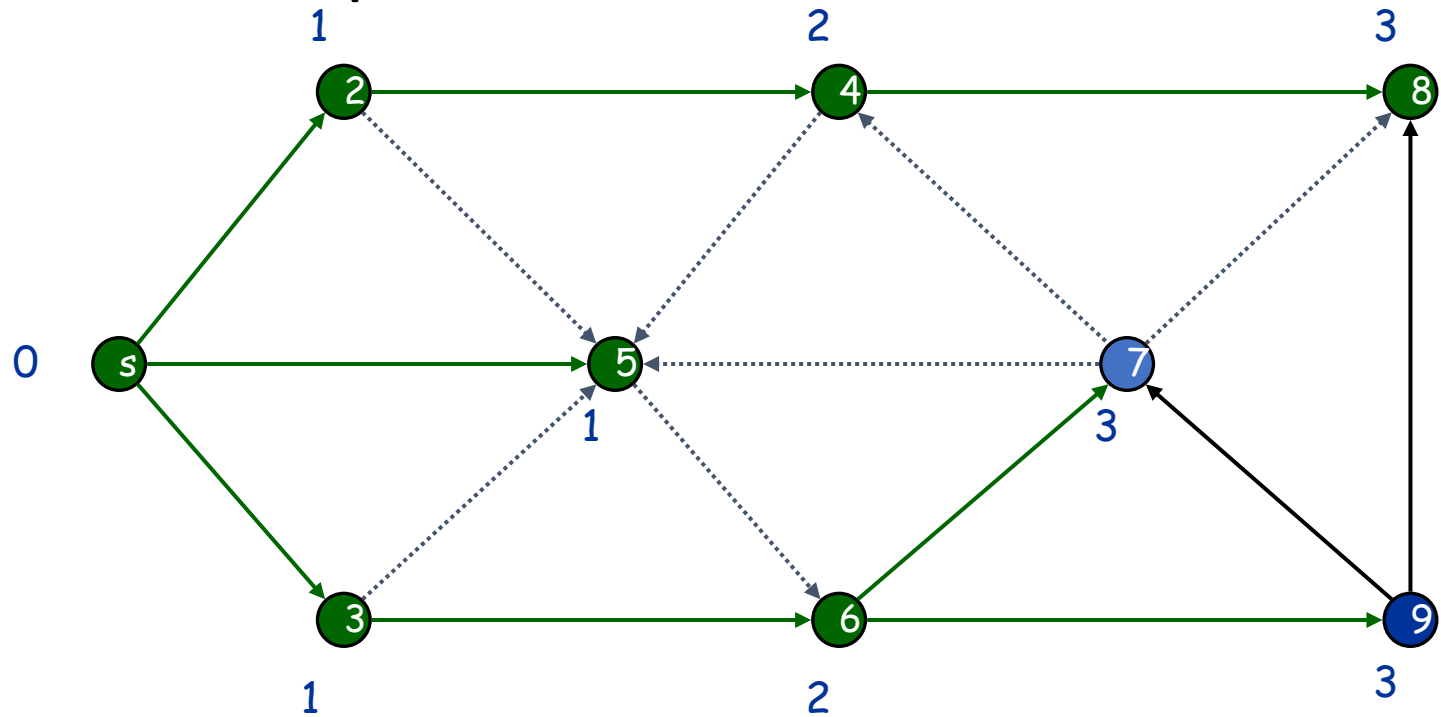
Discovered

Top of queue

Finished

Queue: 7 9

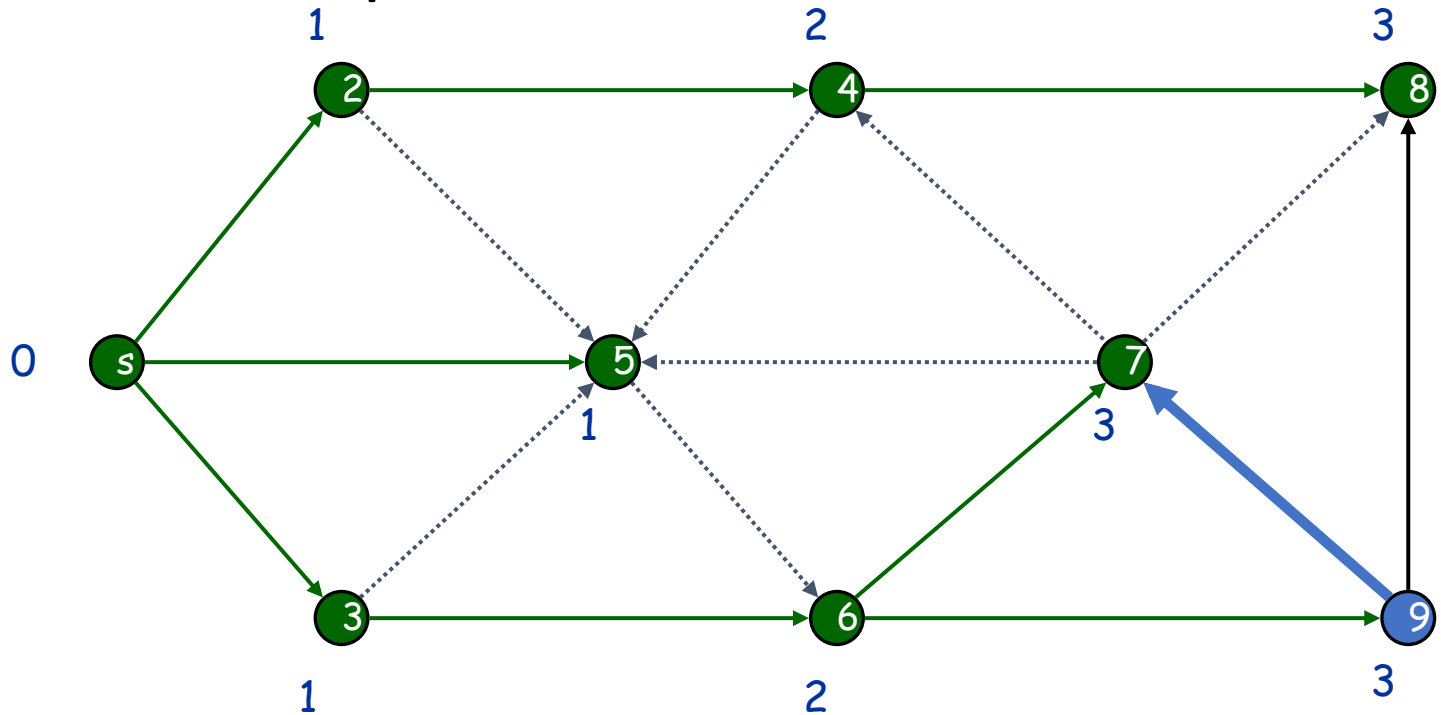
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 7 9

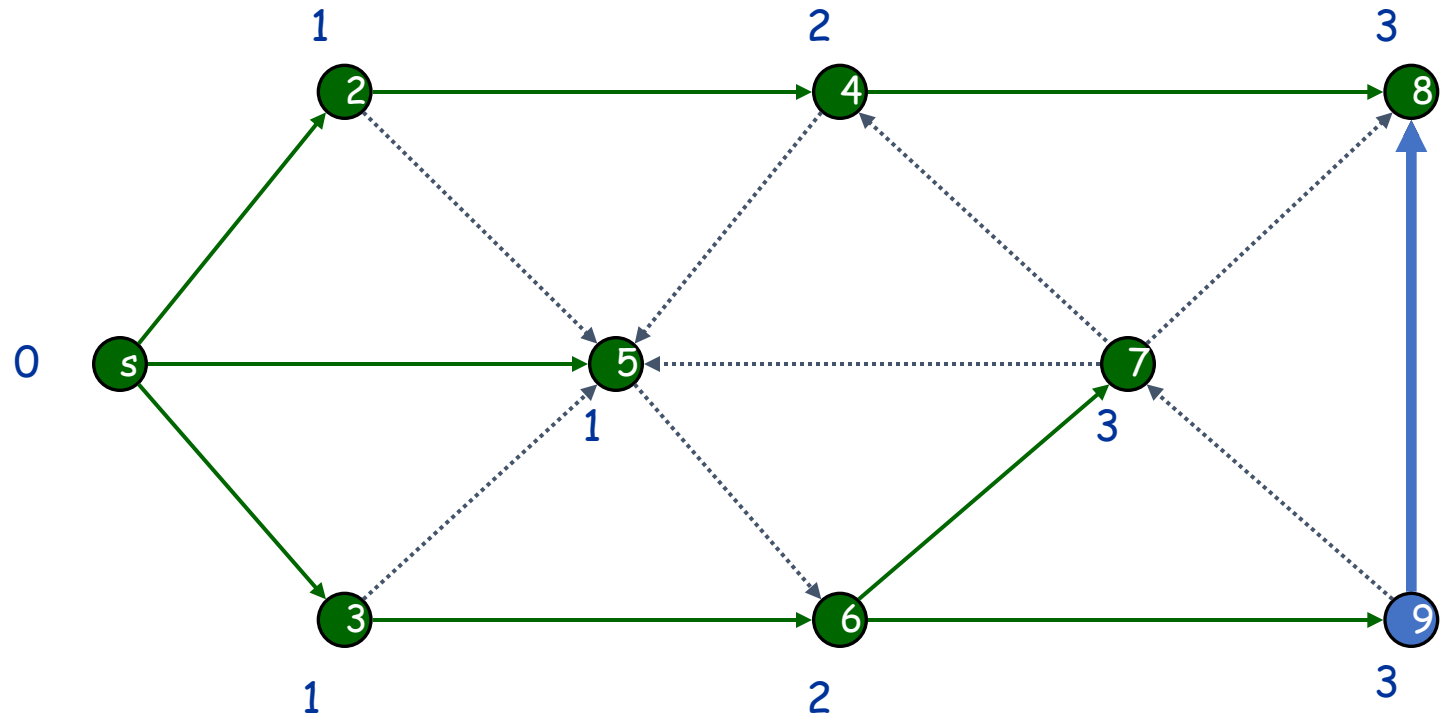
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 9

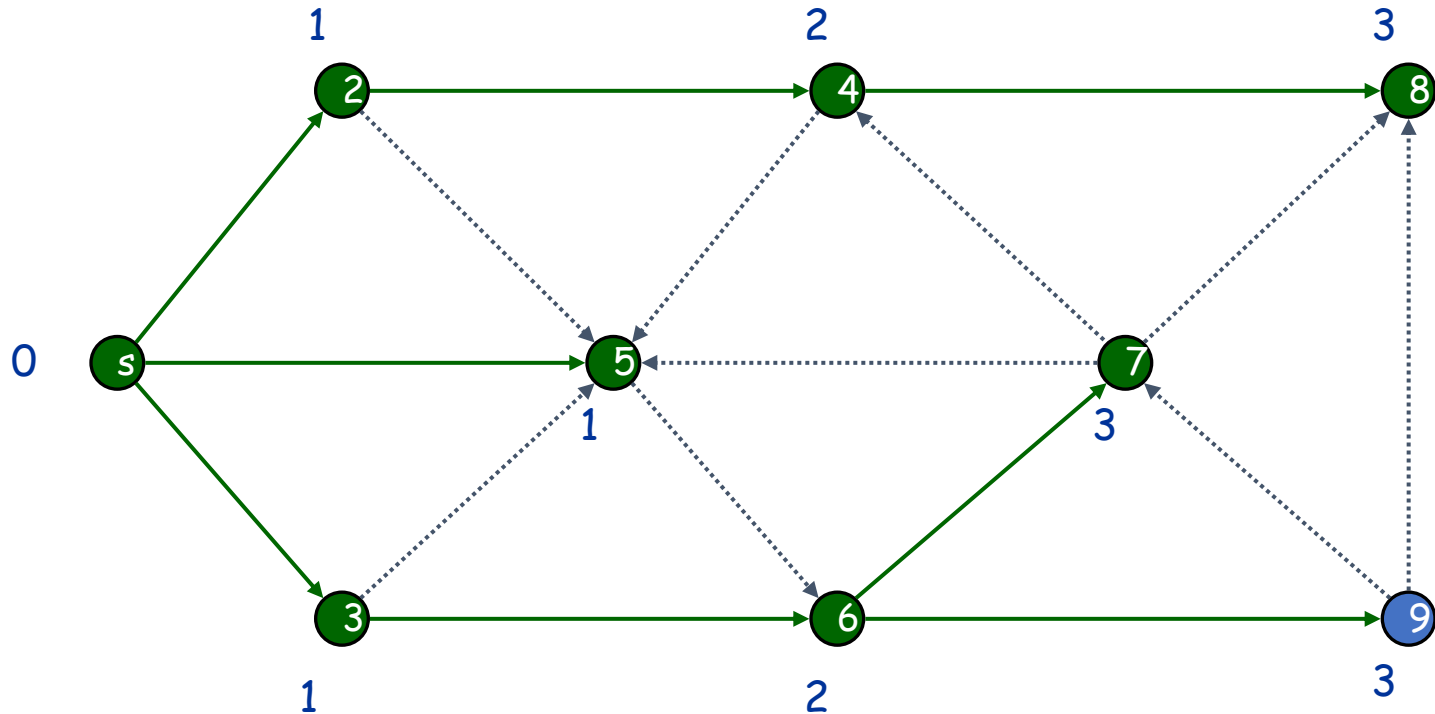
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 9

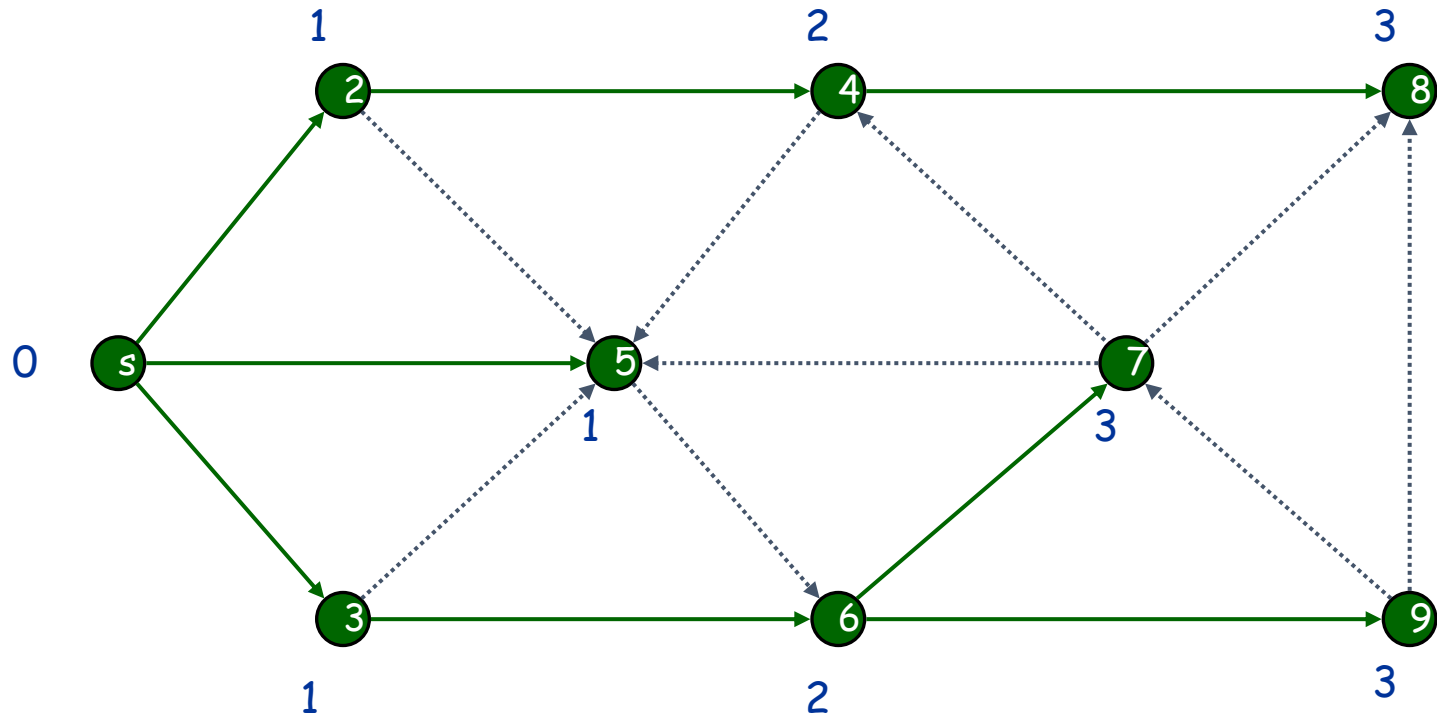
BFS Example



Undiscovered
Discovered
Top of queue
Finished

Queue: 9

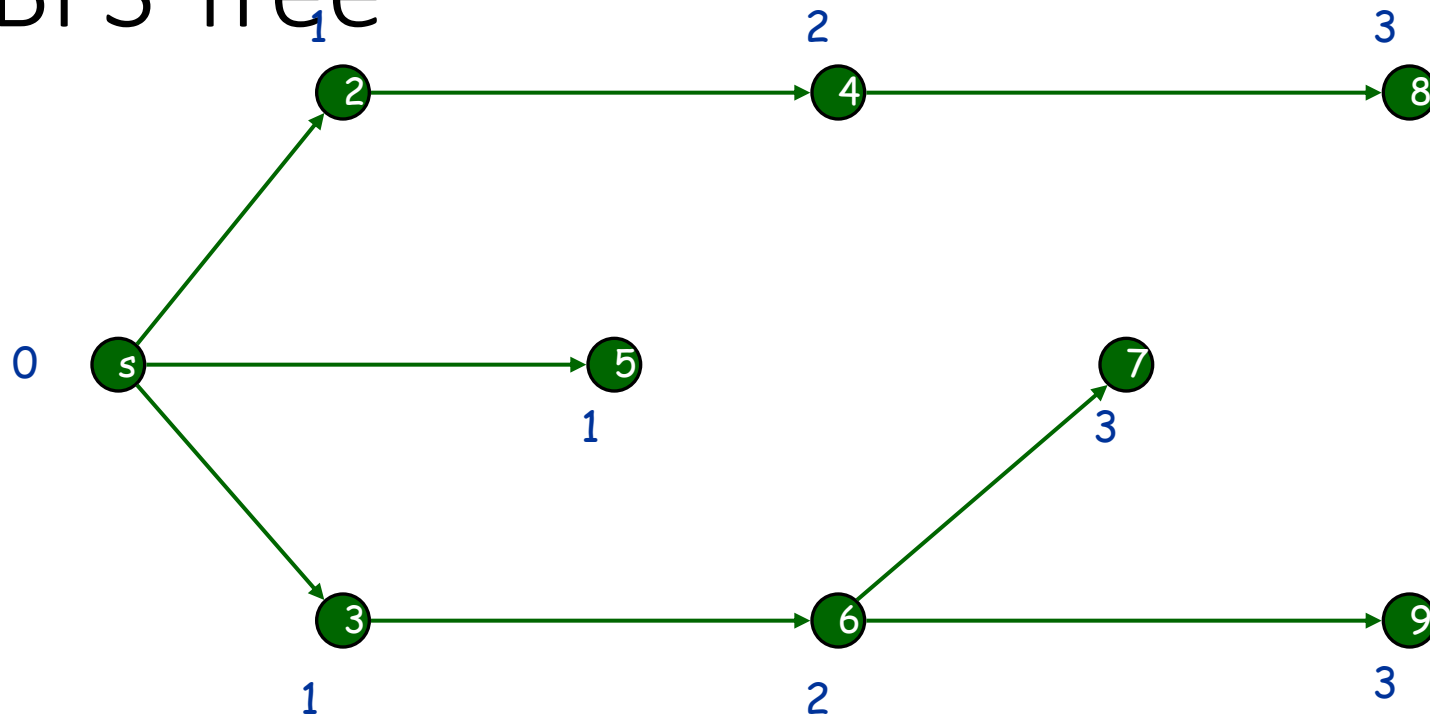
BFS Example



Undiscovered
Discovered
Top of queue
Finished

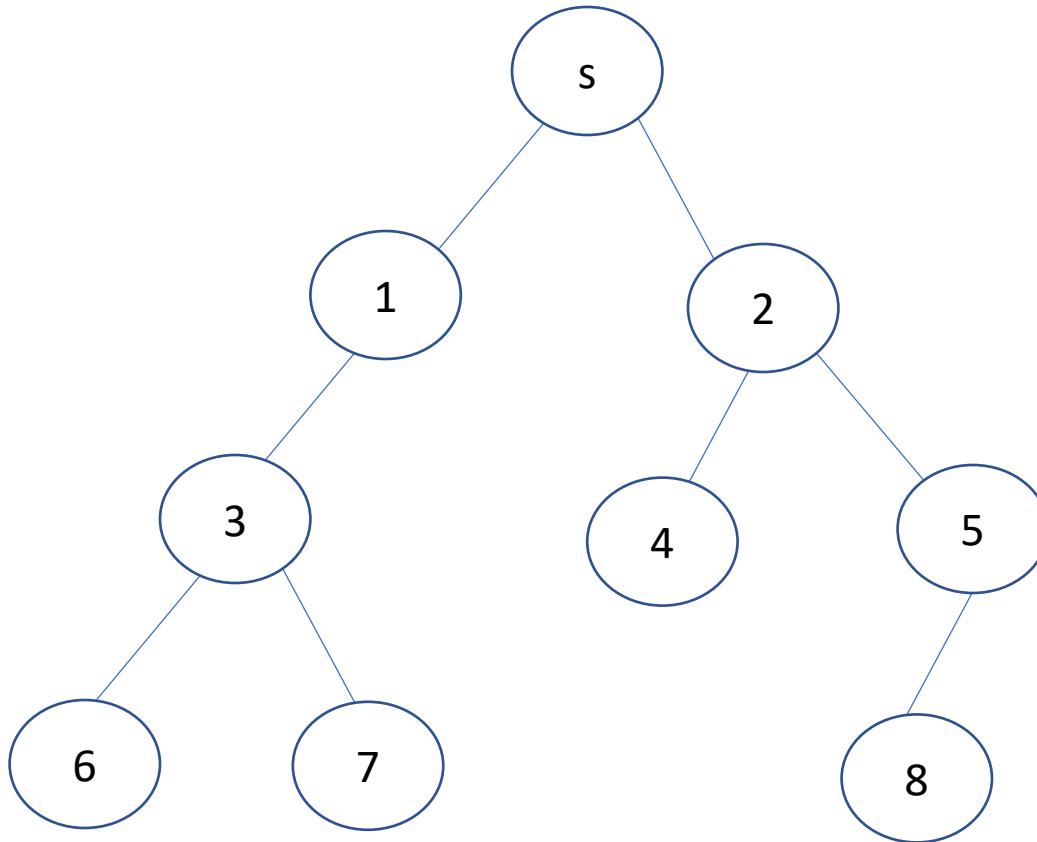
Queue:

BFS Tree

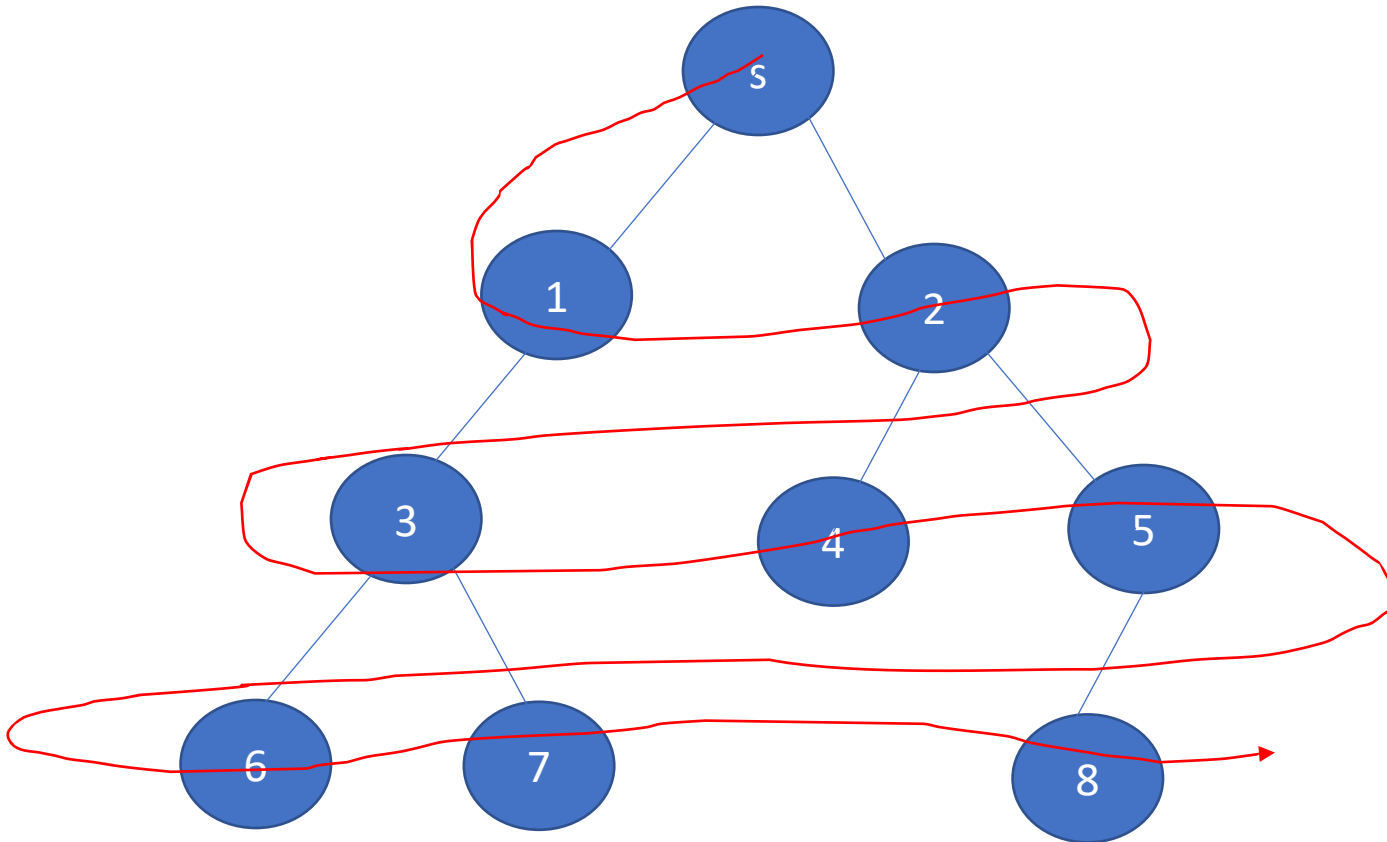


- Starting from s , we visited all (reachable) nodes
- BFS forms a tree rooted at s (BFS Tree)
- For each node x reachable from $s \rightarrow$ we created a shortest path from s to x

BFS on a TREE: Example

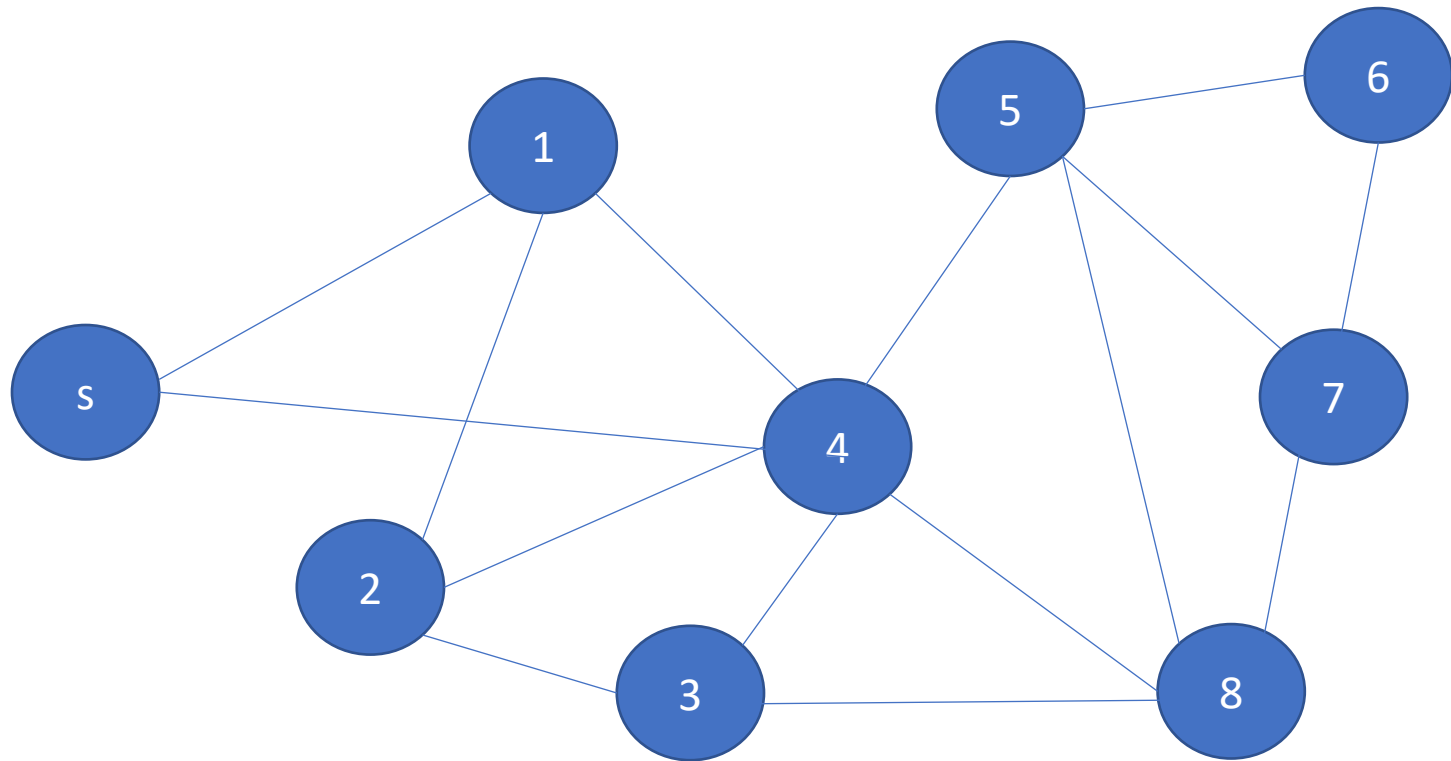


BFS on a TREE: Outcome

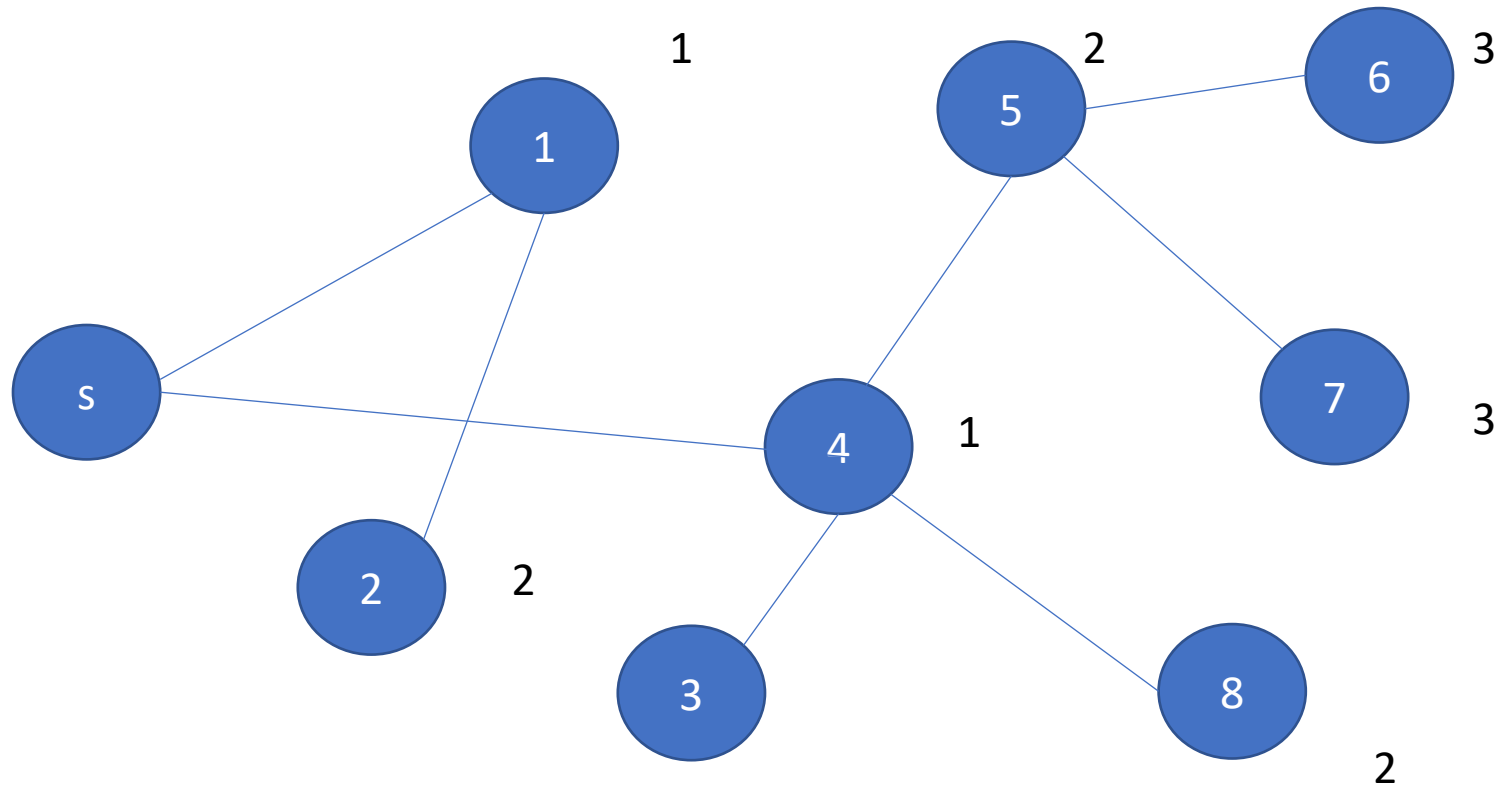


See why it is called Breadth First Search? It goes through the breadth of the graph first

Another BFS Example To Try



Another BFS Example: Outcome



Breadth-First Search

Example problems in which we use BFS

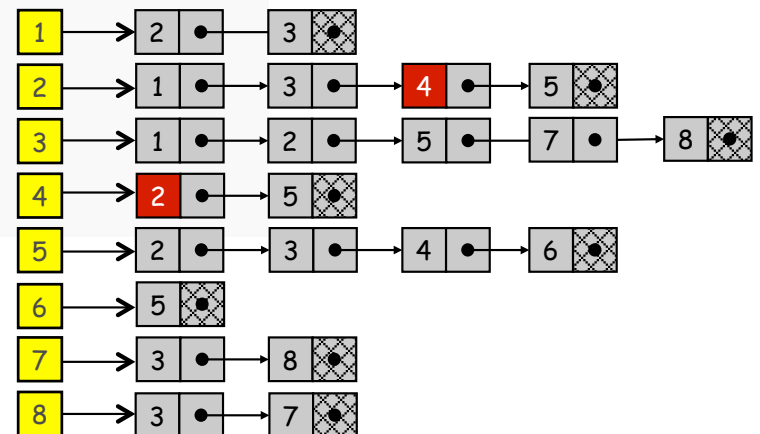
- ◆ Find if node x is reachable from node y
 - ◆ *Start from node y and do BFS*
- ◆ Find the shortest path from node x to node y
 - ◆ *Start from node x and perform BFS*
- ◆ Search for a value v in the graph
 - ◆ *Start from any node and perform BFS*

Always keep in mind whether we talk about undirected graph or directed graph

Breadth-First Search: Algorithm

```

1  procedure BFS( $G, v$ ): ← Start at node  $v$ 
2      create a queue  $Q$ 
3      enqueue  $v$  onto  $Q$ 
4      mark  $v$ 
5      while  $Q$  is not empty:
6           $t \leftarrow Q.dequeue()$ 
7          if  $t$  is what we are looking for: ← Can do any processing on  $t$ 
8              return  $t$ 
9          for all edges  $e$  in  $G.adjacentEdges(t)$  do
12              $u \leftarrow G.adjacentVertex(t, e)$ 
13             if  $u$  is not marked:
14                 mark  $u$ 
15                 enqueue  $u$  onto  $Q$ 
16  return none
    
```



Breadth-First Search: Analysis

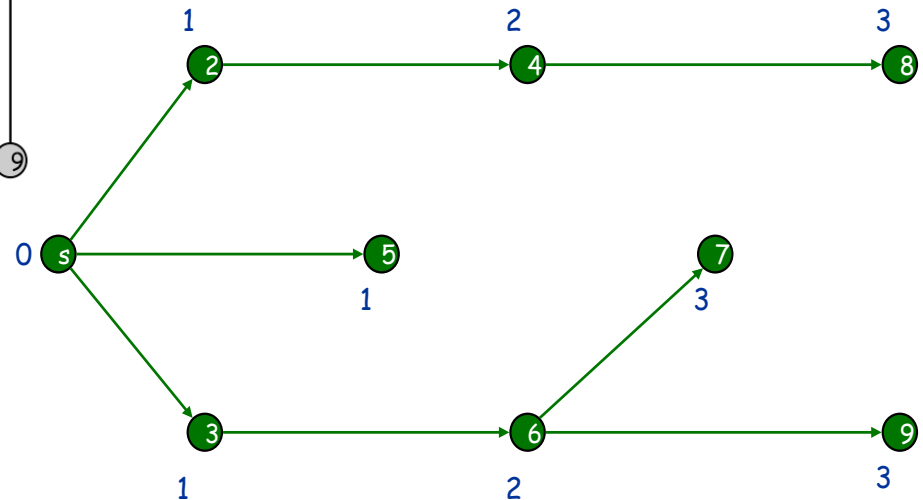
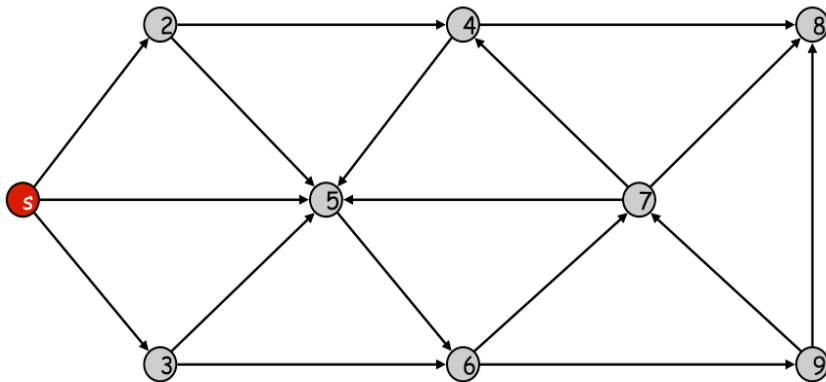
The above implementation of BFS runs in $O(V + E)$ time if the graph is given by its adjacency list.

• Proof

- Each node will be queued only once $\rightarrow O(V)$
- For each node, we visit all its out edges $\rightarrow O(E)$
 - In fact each edge (u,v) is visited twice once from u 's side and once from v 's side
- Total is $O(V + E)$

BFS & Shortest Path

- **BFS can be used to compute the shortest path (minimum number of edges) from source s and any reachable nodes v**
 - Maintain a counter for each node
 - When a node x is first visited from parent $y \rightarrow x.\text{counter} = y.\text{counter} + 1$





That's all Folks!
Any Question?