

CSCI 256 Lecture 5:

Graphs & BFS

Shikha Singh

Announcements

- Hand in Exercise #4, No exercise for Wed 9/23
- Focus on Assignment 1 due Wed (9/23) at 10 pm
- Must use LaTeX and Overleaf template
- Joint write up submitted to Gradescope: add your partner
- Working on grading:
 - Graded exercises will be released on Gradescope
 - Graded Assignment 0 as well

Group Members

 Add or remove group members for this submission.

Your instructor has allowed you to submit as a group of up to any number of people. You can change the group below. Students added or removed will be notified via email.

Student	Remove
Katy Student (Submitter)	
Byron Washington	

Add Student

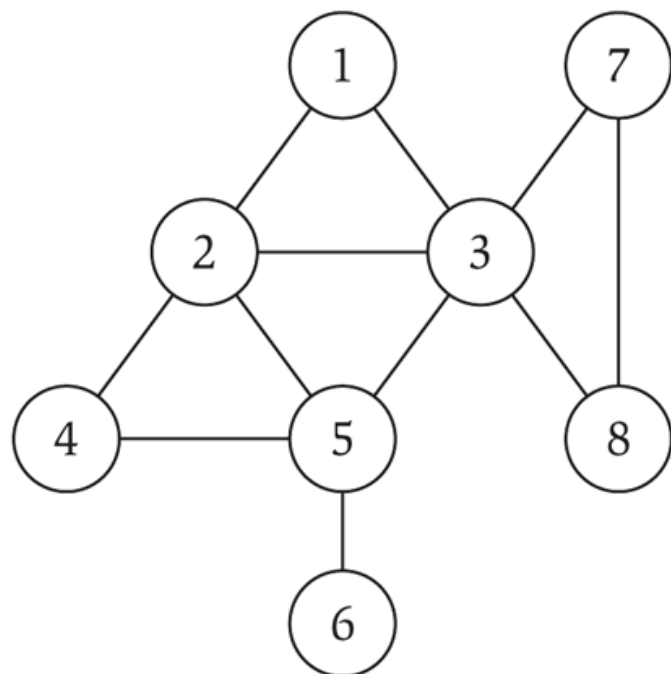
CancelAdd

Review: Undirected Graphs

An undirected graph $G = (V, E)$

- V is the set of nodes, E is the set of edges
- Captures pairwise relations between objects
- Graph size parameters: $n = |V|$, $m = |E|$

Sometimes we consider weighted graphs, where each edge e has a weight $w(e)$



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

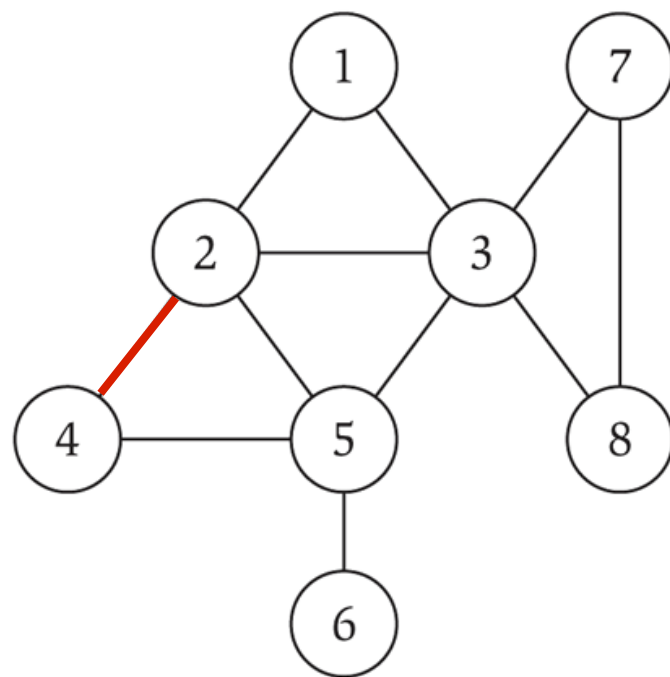
$$E = \{ 1-2, 1-3, 2-3, 2-4, 2-5, 3-4, 3-5, 3-7, 3-8, 4-5, 5-6, 7-8 \}$$

$$m = 11, n = 8$$

Graph Data Structures

Adjacency matrix.

- n -by- n matrix where $A[u][v] = 1$ if $(u, v) \in E$
- Space $O(n^2)$
- Checking if $(u, v) \in E$ takes $O(1)$ time

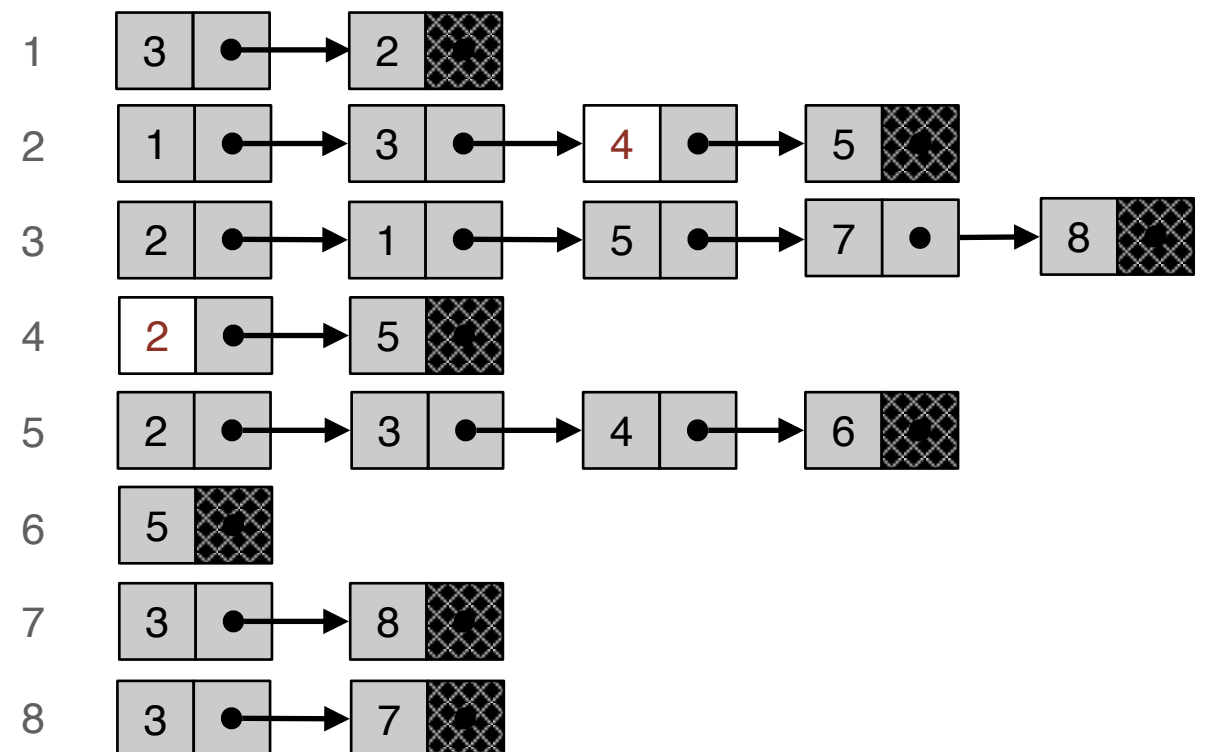
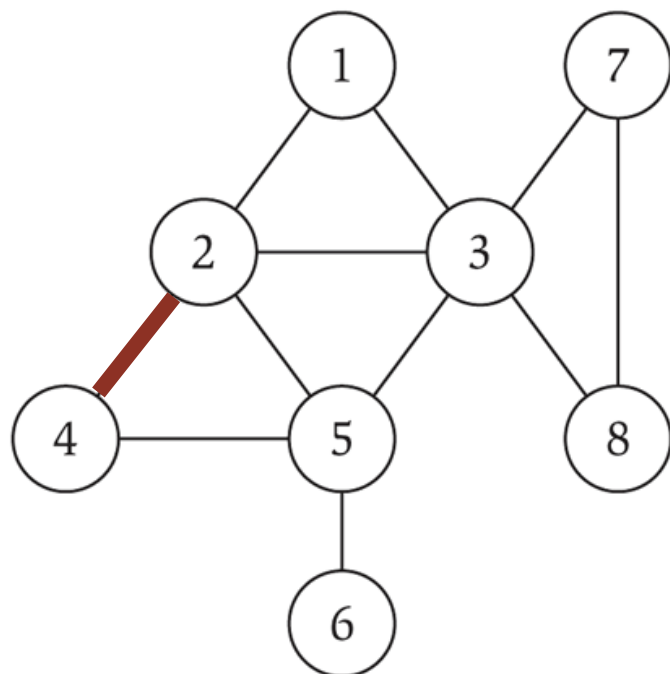


	1	2	3	4	5	6	7	8
1	1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Graph Data Structures

Adjacency list.

- Array of lists, where each list represents the neighbors of a given node
- Space $O(n + m)$
- Checking if $(u, v) \in E$ takes $O(\text{degree}(u))$ time



Graph Terminology

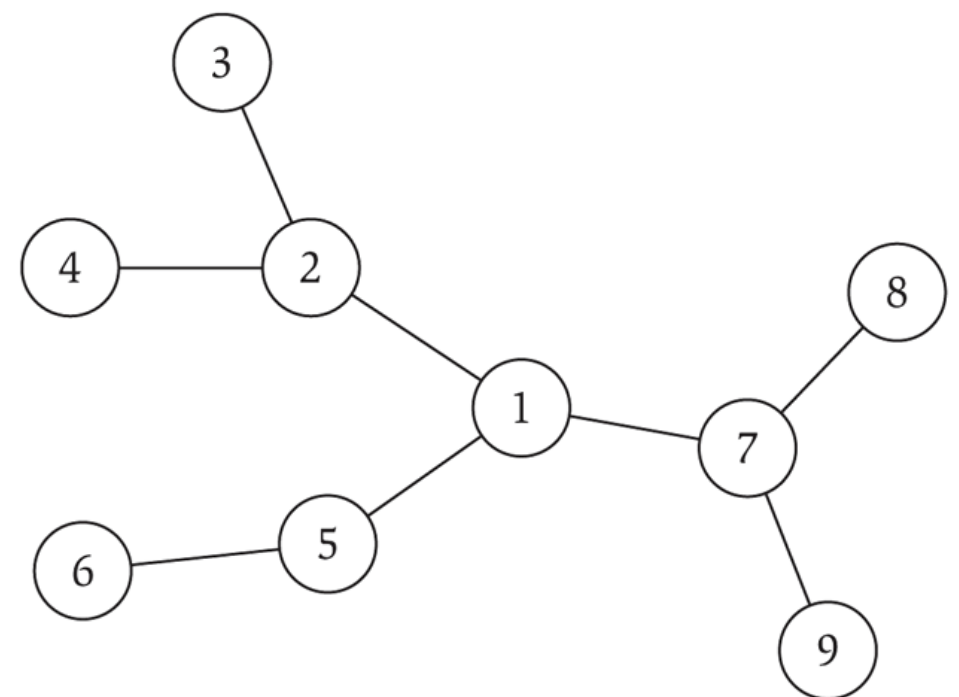
- A **path** in an undirected graph $G = (V, E)$ is a sequence of nodes $u_1, u_2, \dots, u_k$ such that every pair $(u_{i-1}, u_i) \in E$.
- A path is **simple** if all nodes are distinct
- The **length of a path** is the number of edges on the path
- An undirected graph is **connected** if for every pair of nodes u and v , there is a path between u and v
- A **cycle** is path $u_1, u_2, \dots, u_k$ where $u_1 = u_k$ ($k \geq 2$)
- A cycle is simple if all internal nodes are distinct

Trees

- An undirected graph is a tree if it is connected and does not contain a cycle

Lemma. Let G be an undirected graph with n nodes. Then any two of these conditions imply the third

- G is connected
- G does not contain a cycle
- G has $n - 1$ edges

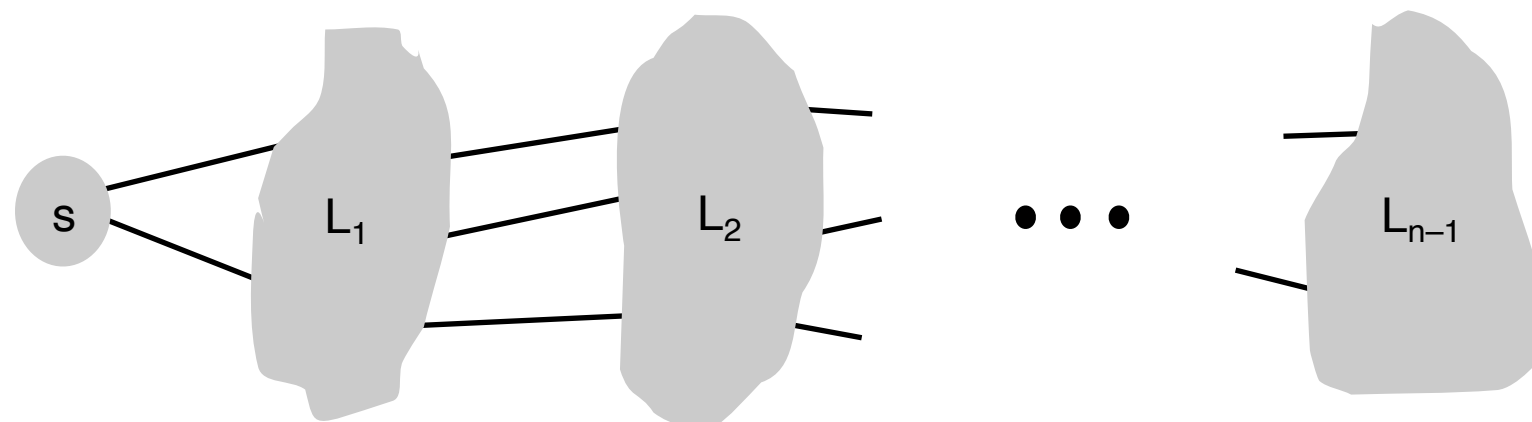


Graph Traversals

- **Connectivity.** How do we verify if a graph is connected?
- **Path.** Given $s, t \in V$, is there a path between them?
- Determined by “traversing the graph”
- Two classic graph traversal algorithms:
 - Breadth-first search (BFS)
 - Depth-first search (DFS)
 - Both are building blocks of many graph algorithms

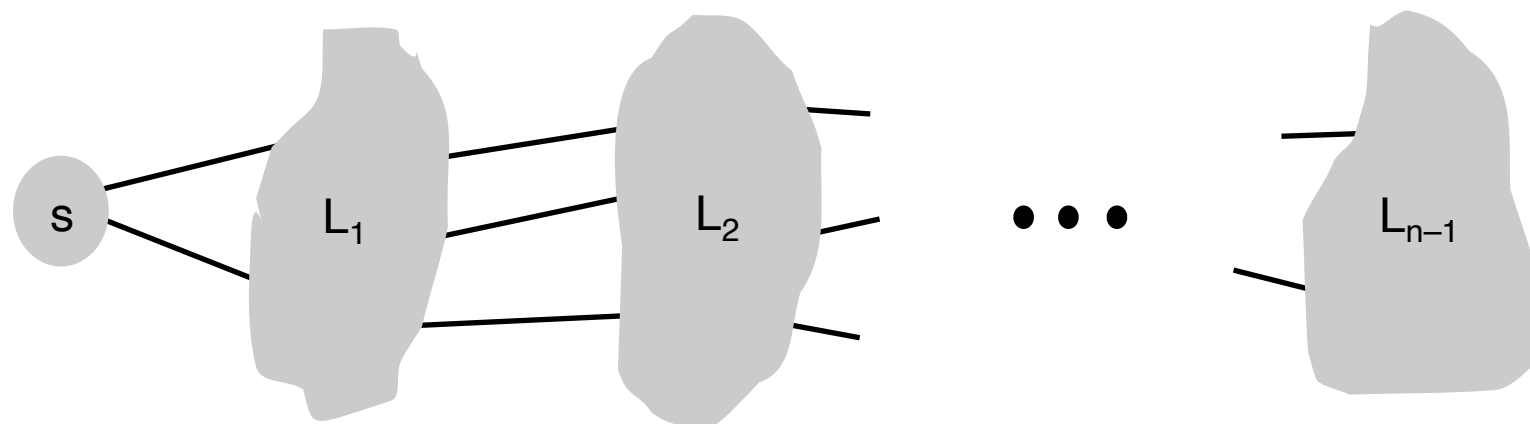
Breadth-first Search

- Search outwards from some vertex s and explore in all directions “one level at a time”
- Start with s , then explore neighbors of s
 - Then explore neighbors of neighbors of s
 - Then explore neighbors of neighbors of neighbors of s
 - .. and so on

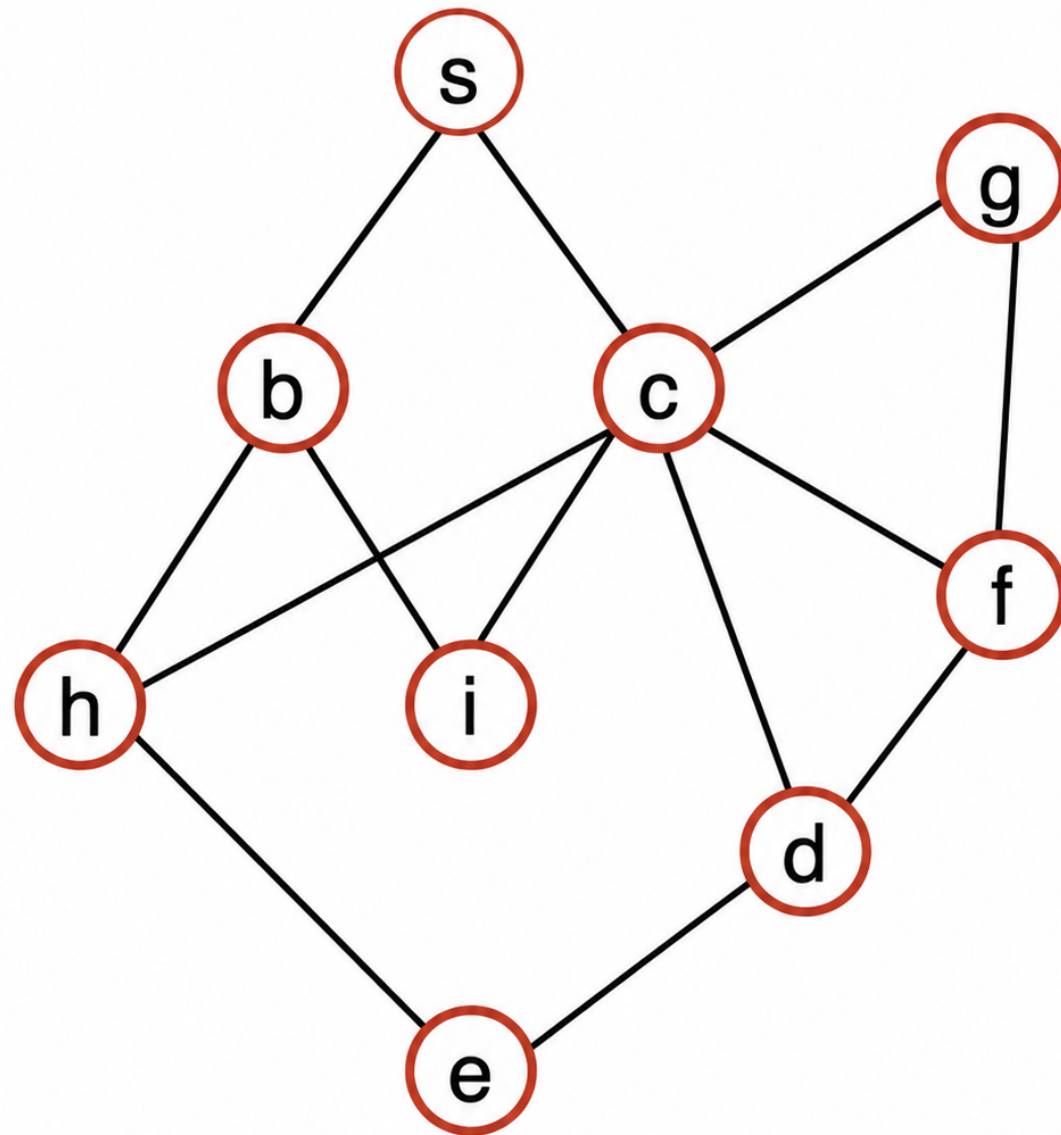


Breadth-first Search

- Search outwards from root s in all directions “one layer at a time”
- Initialize $L_0 = \{s\}$
 - $L_1 =$ all neighbors of L_0
 - $L_2 =$ all nodes adjacent to a node in L_1 that are not in $\{L_0, L_1\}$
 - $L_{i+1} =$ all nodes adjacent to a node in L_i that are not in any earlier layer



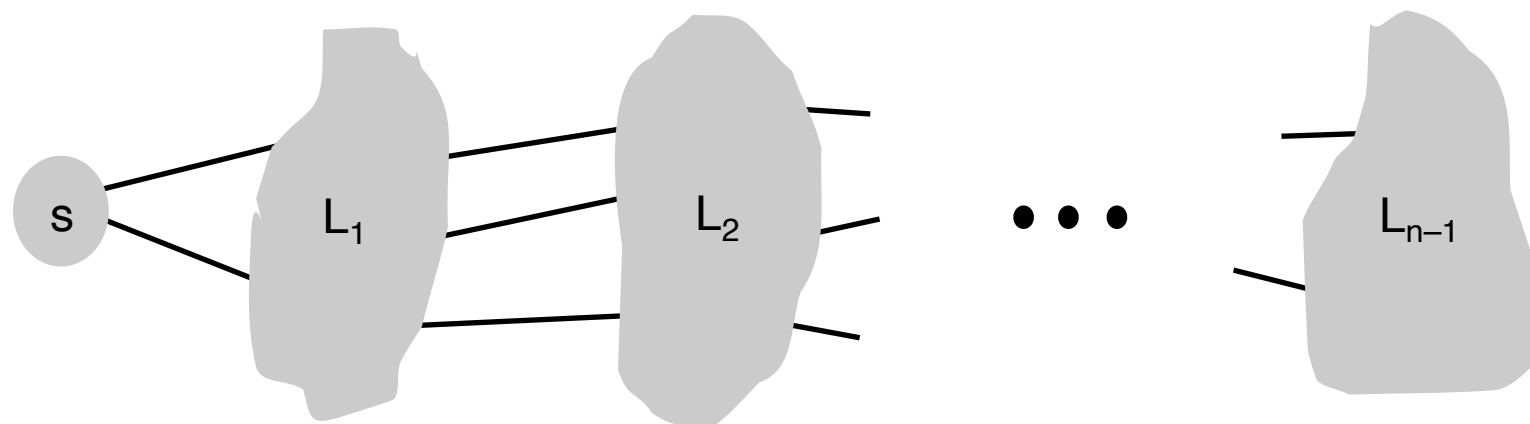
BFS Example



BFS Levels

- Initialize $i = 0$ and $L_0 = \{s\}$, mark s as visited
- While there exists a node u in L_i with an unvisited neighbor v
 - Mark v as visited and add v to L_{i+1}
- Update level $i \leftarrow i + 1$

All nodes at level L_i
explored before L_{i+1}



BFS Implementation: Levels

BFS(s):

Set $\text{Discovered}[s] = \text{true}$ and $\text{Discovered}[v] = \text{false}$ for all other v

Initialize $L[0]$ to consist of the single element s

Set the layer counter $i = 0$

Set the current BFS tree $T = \emptyset$

While $L[i]$ is not empty

 Initialize an empty list $L[i+1]$

 For each node $u \in L[i]$

 Consider each edge (u, v) incident to u

 If $\text{Discovered}[v] = \text{false}$ then

 Set $\text{Discovered}[v] = \text{true}$

 Add edge (u, v) to the tree T

 Add v to the list $L[i+1]$

 Endif

 Endfor

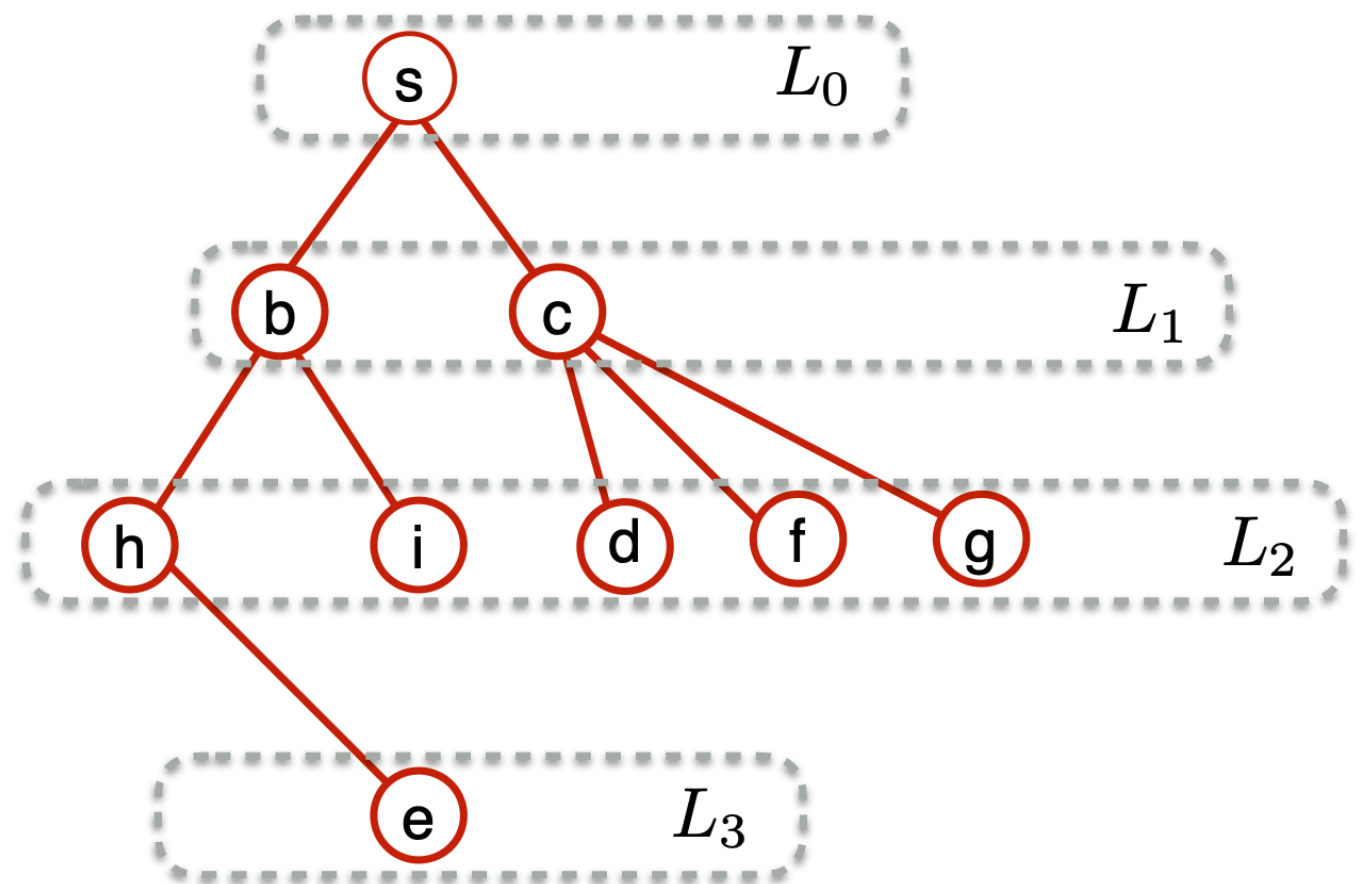
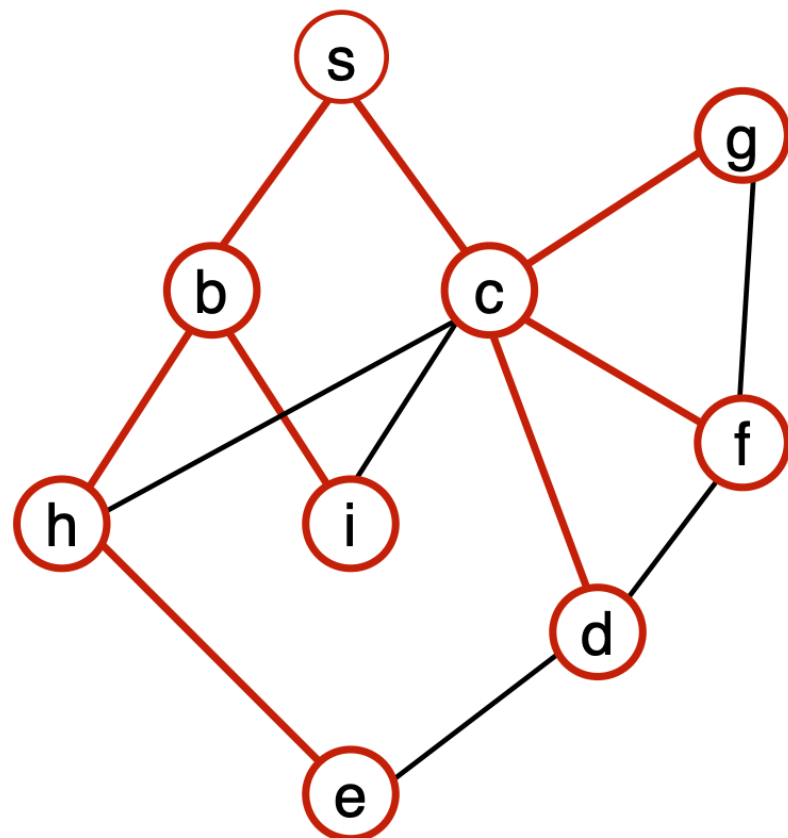
 Increment the layer counter i by one

Endwhile

All nodes at level L_i
explored before L_{i+1}

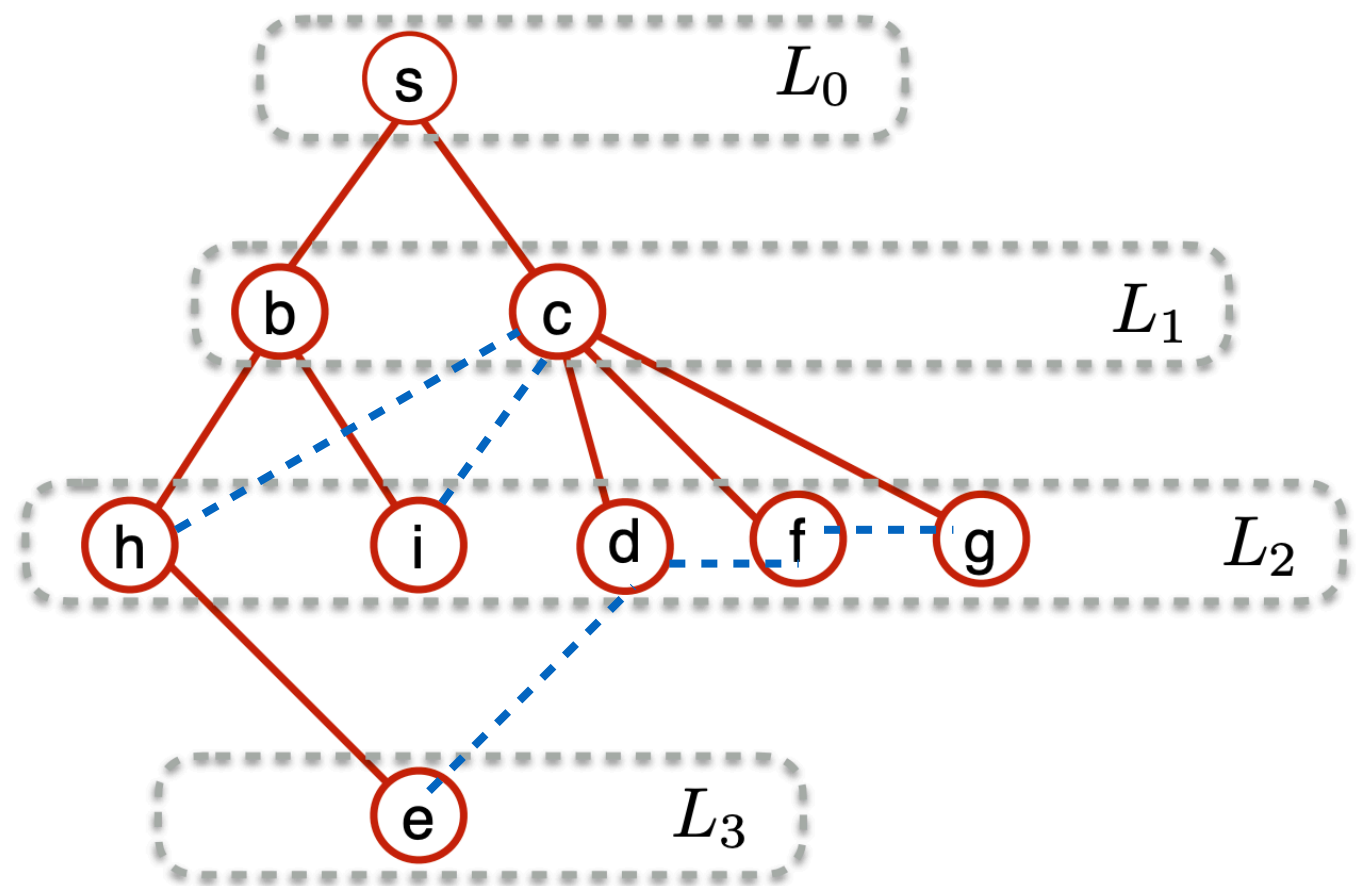
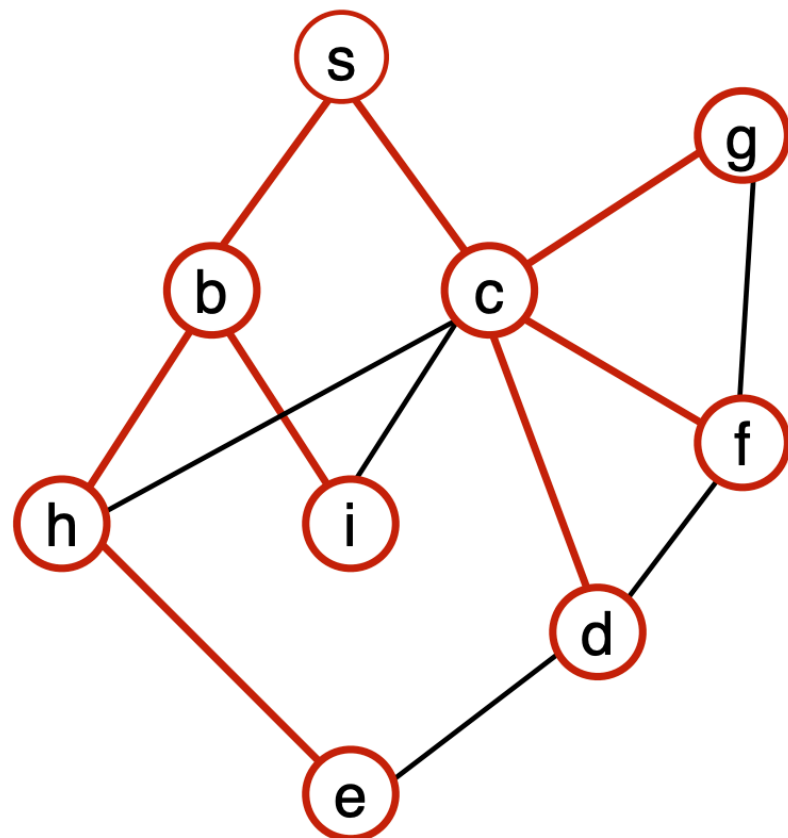
BFS Tree Property

- **Definition.** Let T be the tree formed by a BFS search such that each node's parent is the node that first discovered it.



BFS Tree Property

- **Theorem.** Let (x, y) be any edge of an undirected graph G . Then, the BFS levels of x and y differ by at most 1.

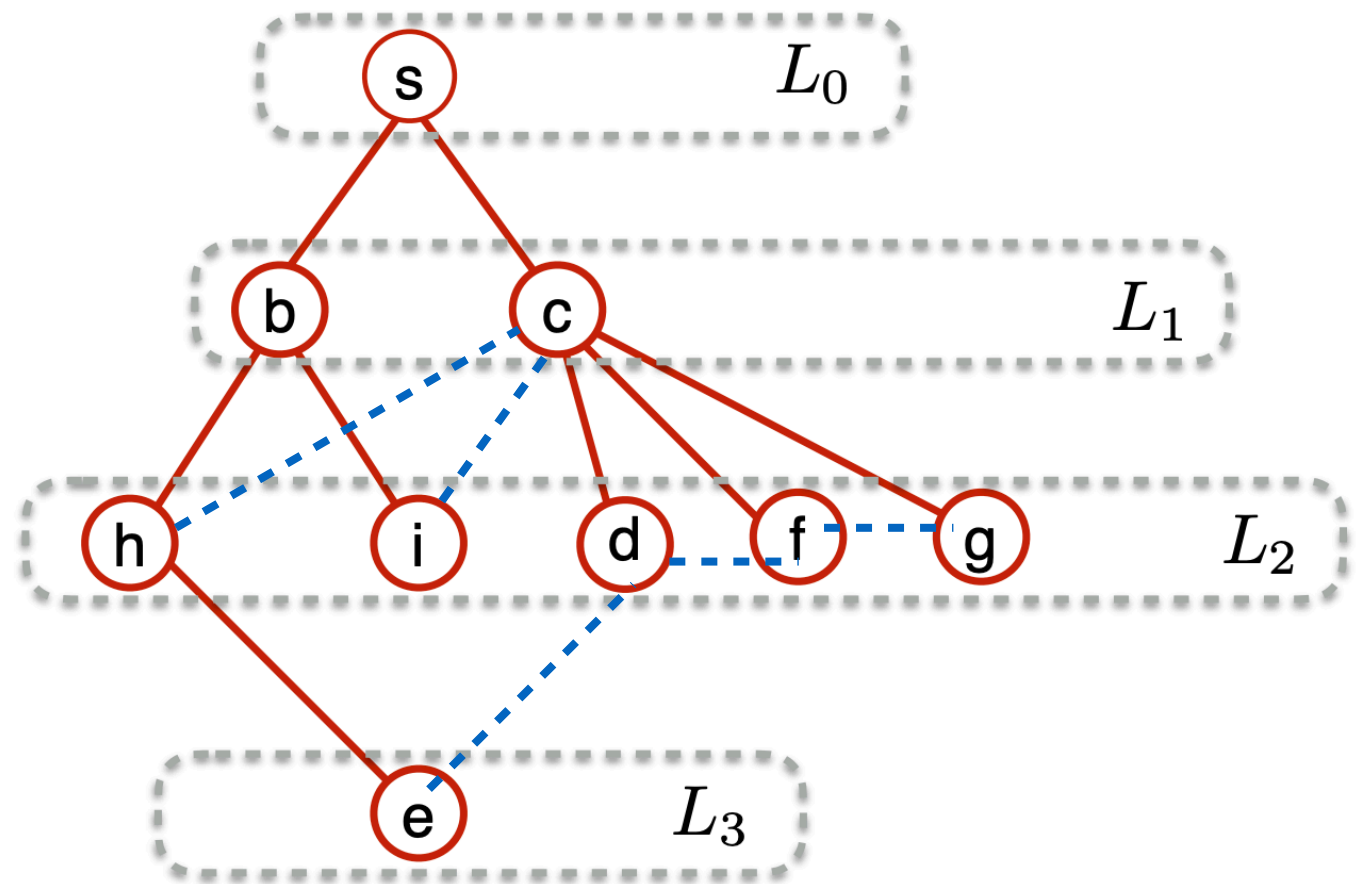
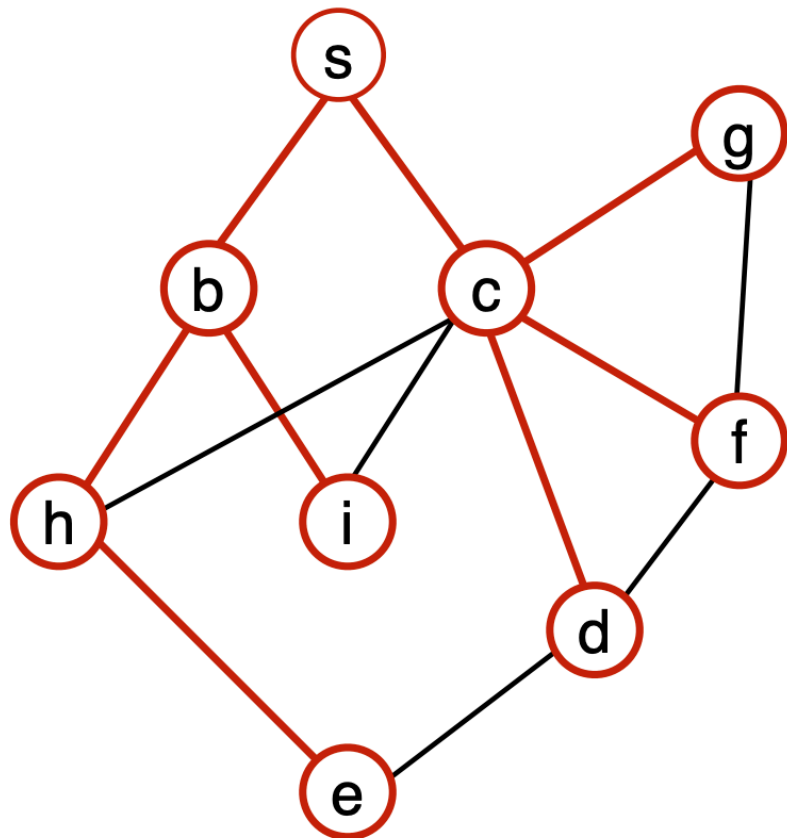


BFS Tree Property

- **Theorem.** Let (x, y) be any edge of an undirected graph G . Then, the BFS levels of x and y differ by at most 1.
- Proof. If x and y are in the same level, the property holds
- Suppose now that x and y are at different levels
- Suppose x is at level i and y is at some level j
- (Without loss of generality) Suppose $i < j$, since BFS explores nodes at level i first and y is not at any level $1, \dots, i$ and y is a neighbor of x and is unvisited, y will be added to level $i + 1$
 - Thus, their levels differ by at most 1

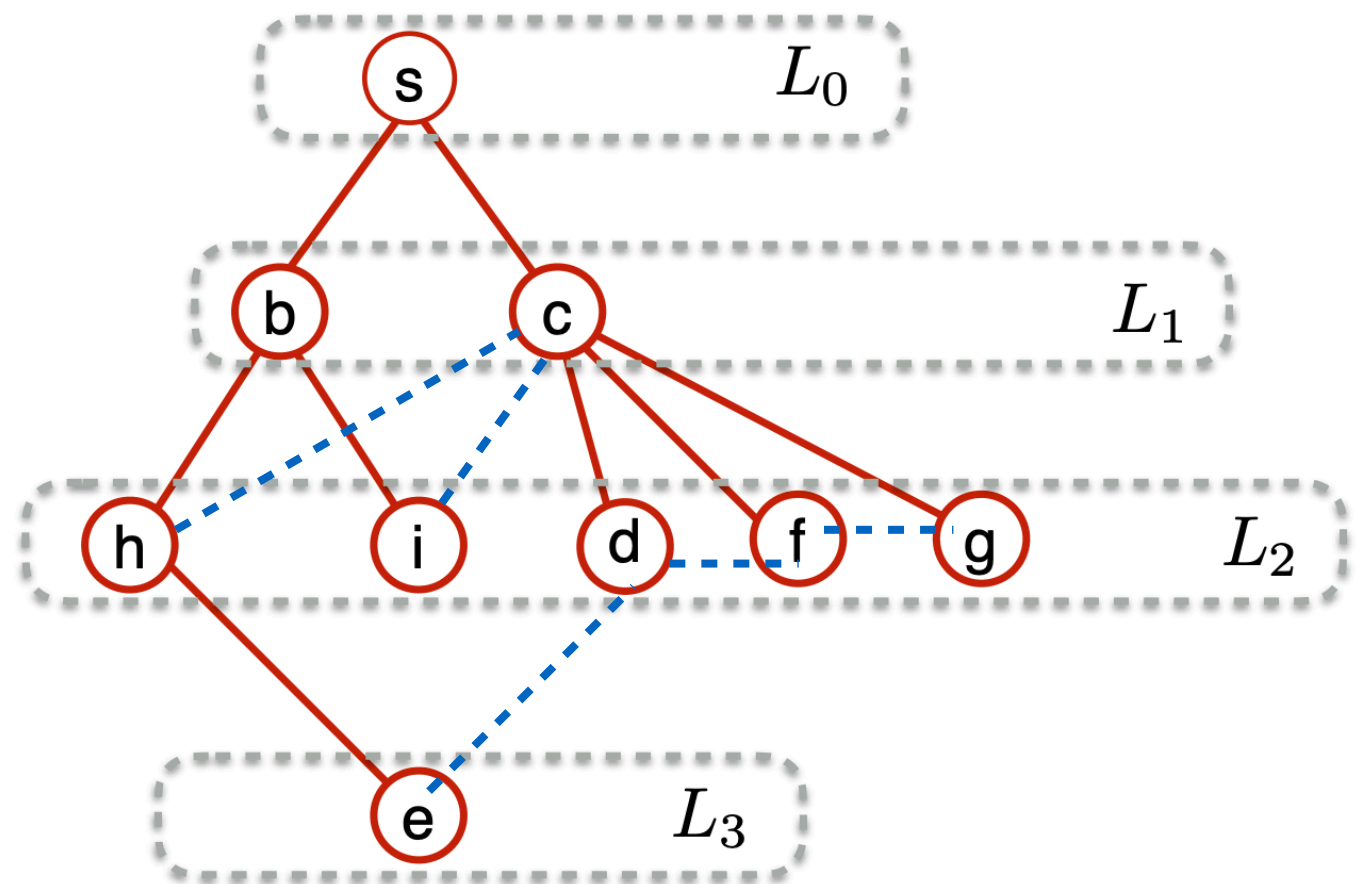
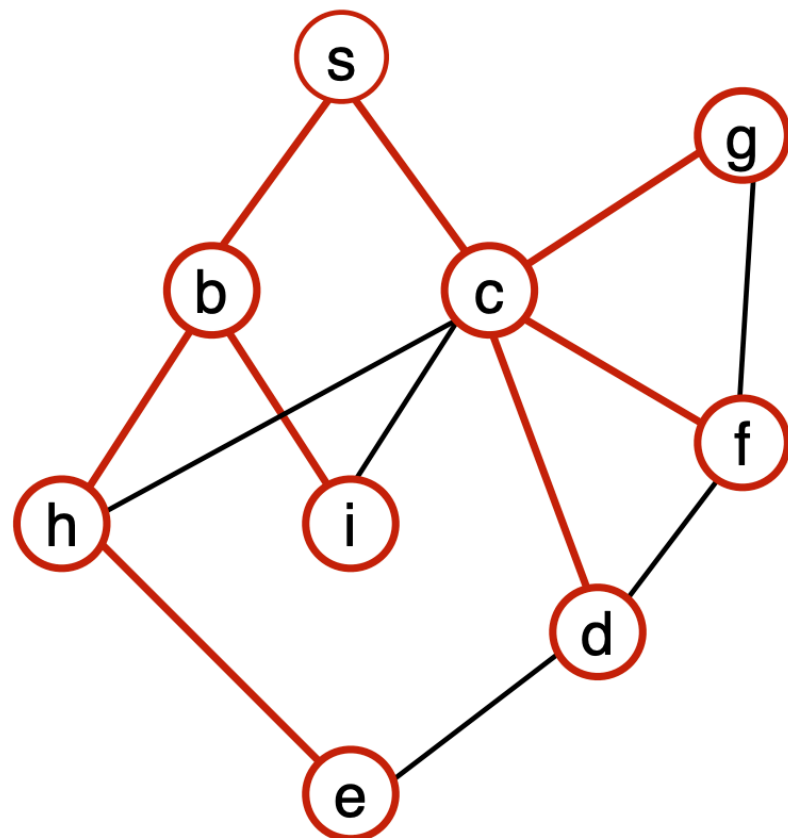
BFS Tree: Shortest Path

- **Property.** For any node v , *the length of the shortest path* connecting s to v is equal to the level of v in BFS.



BFS Tree: Shortest Path

- **Proof.** (By Contradiction) Suppose there is a shorter path from s to v . That path would have to bypass at least one level.
 - Cannot happen since neighbors are in same or adjacent levels

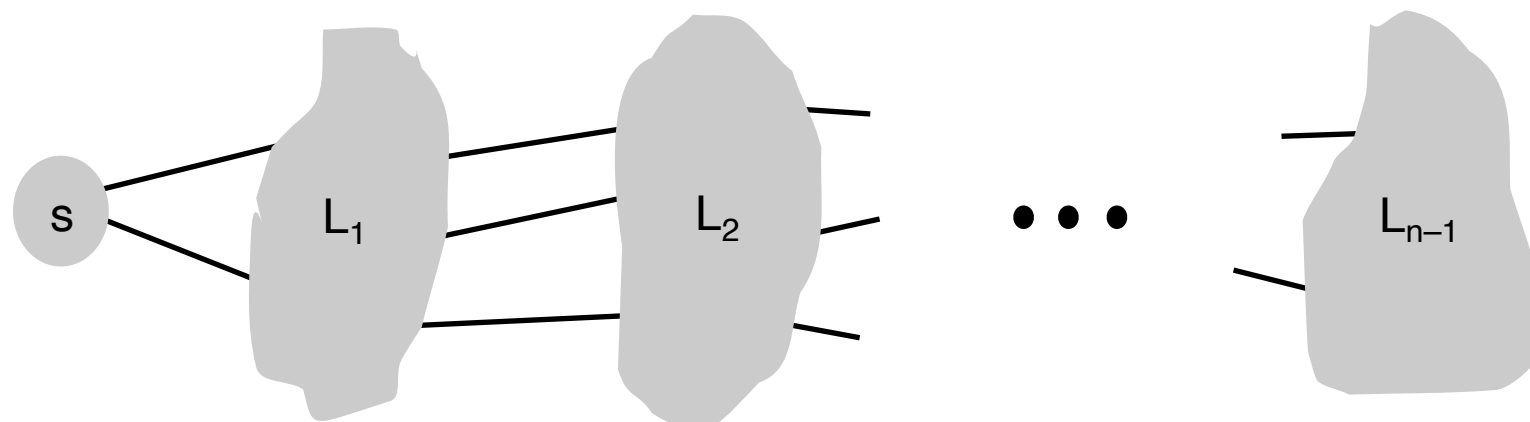


BFS Implementation

- Nodes that we have not seen yet
- Nodes that we have visited
- Nodes that have been “explored” (visited all its neighbors as well)

How to represent G ?

All nodes at level L_i explored before L_{i+1} :
which data structure to use?



BFS Implementation

BFS (G, s):

Store G using an adjacency list

for all $v \neq s$: set $L[v] = \infty$

Set $L[s] = 0$

Put s in the **queue** Q

While Q is not empty

 Extract v from Q

 For each edge (v, w) :

 if $L[w] = \infty$ then // w is unmarked

 Put w into the **queue** Q

$L[w] = L[v] + 1$

Why does queue ensure we visit nodes level-by-level?

BFS Using a Queue

- **Claim.** When BFS is about to ***begin exploring*** a vertex at level L_i , then all vertices in the queue must have level i
- Can prove this by induction

BFS Using a Queue

- **Claim.** When BFS is about to begin **exploring** a vertex at level L_i , then all vertices in the queue must have level i
- **Proof.** Base case. $i = 0$: queue only contains s
- Suppose this holds until level i : as BFS explores these vertices all their neighbors (with level $i + 1$ are added to the back of the queue)
- Since all nodes with level i are before $i + 1$, they all have be explored before BFS begins exploring any node at level $i + 1$ (FIFO property)

BFS Analysis

- Inserting and extracting an edge from a queue
 - $O(1)$ time
- For each marked node v , we run the for loop for its edges: $O(n)$
- Overall running time?
 - Easy to prove $O(n^2)$ time
- Can improve the analysis to $O(n + m)$
 - Node u has $\text{degree}(u)$ incident edges (u, v)
 - Total time processing edges: $\sum_{u \in V} \text{degree}(u) = 2m$

each edge (u, v) is counted exactly twice
in sum: once in $\text{degree}(u)$ and once in $\text{degree}(v)$