

Data Structures and Algorithms

Prepared by Prof. Shereen Taie

IT251
Information Technology Department

Lecture 8



Data Structures and Algorithms

Lecture 8

IT251

Information Technology Department

Prepared by
Prof. Shereen Taie

Learning Objectives:

- **Graphs & Terminology**
- **WHAT ARE GRAPHS?**
- **GRAPH EXAMPLES**
- **KINDS OF GRAPHS**
- **GRAPH REPRESENTATIONS**
- **Sparse vs. Dense Graphs**
- **BFS (Breadth-first search)**

Graphs Terminology

Chapter

04

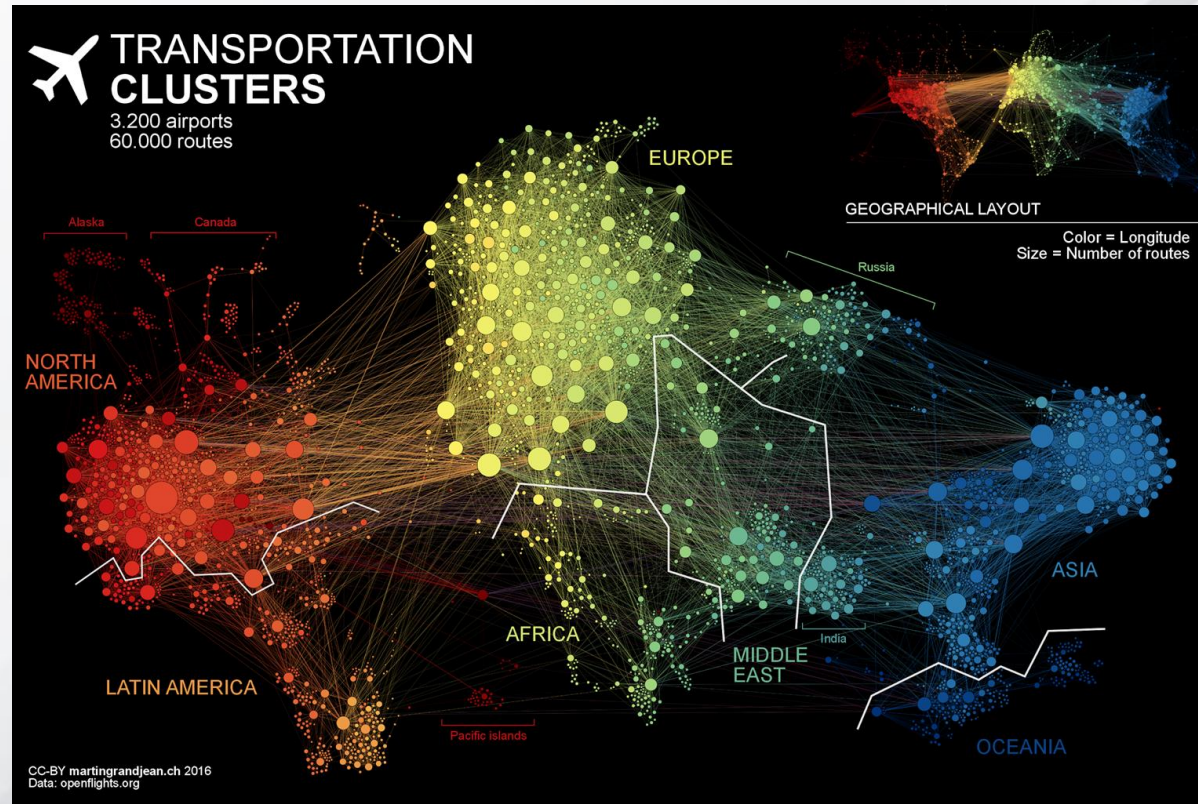
WHAT ARE GRAPHS?

Chapter

04

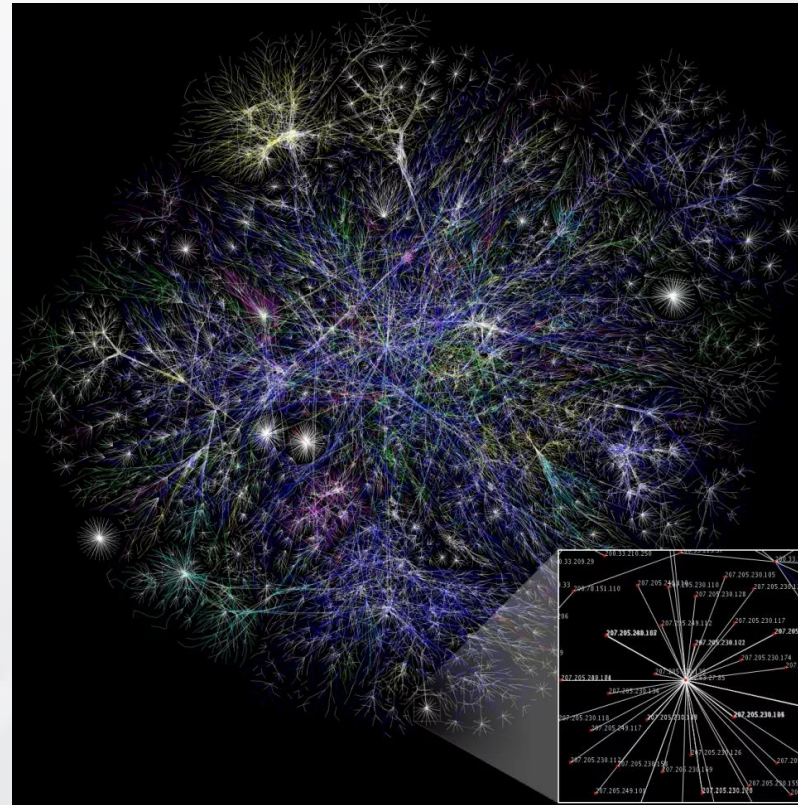
GRAPH EXAMPLES

Each “node” is an airport, and flight routes are represented by the “edge” in between them



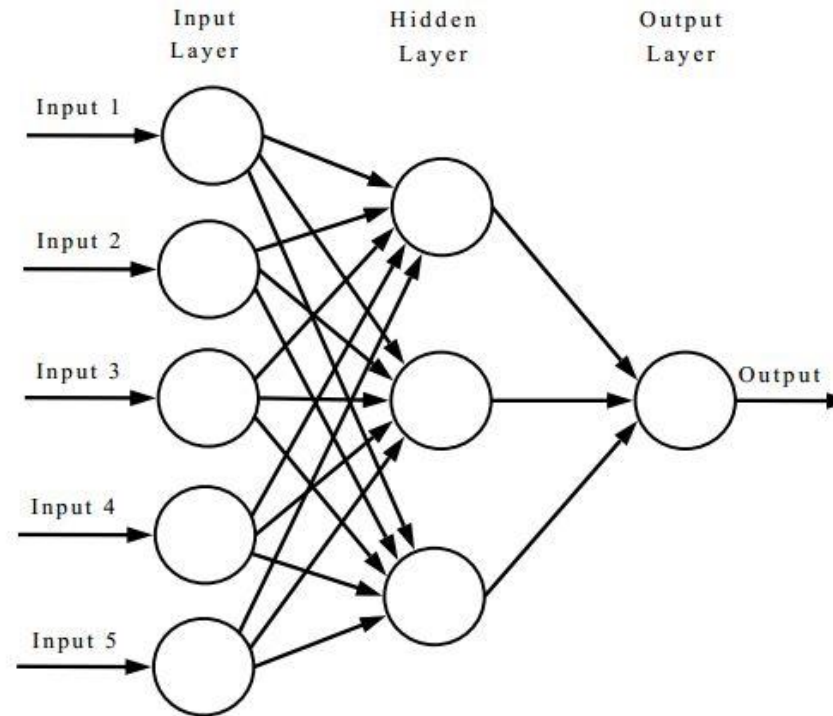
GRAPH EXAMPLES

Partial graph of the Internet (in 2005), where each “node” is an IP address, and the “edges” between them reveal connectivity delays (shorter lines = closer IP addresses)



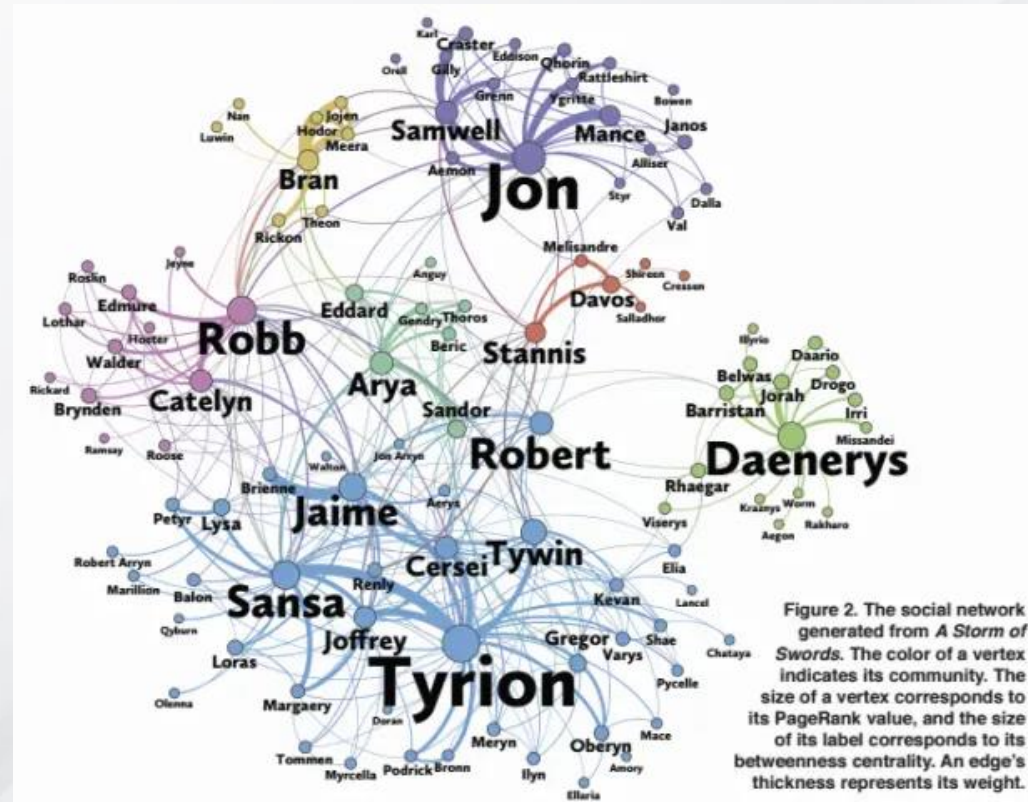
GRAPH EXAMPLES

Neural networks! Each “node” represents a module of the neural network, and “edge” represent output/input relationships



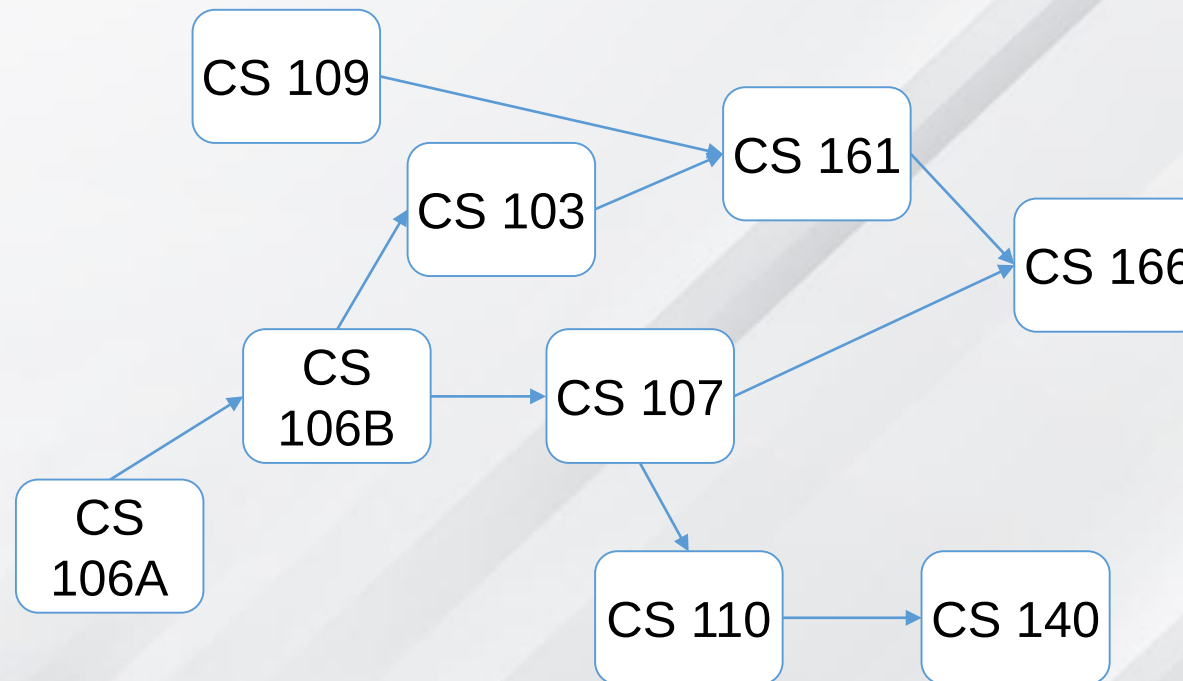
GRAPH EXAMPLES

Graph of characters in the third book of Game of Thrones, where each “node” is a character, and “edge” reveal frequency of interaction (i.e. 2 names appearing within 15 words of one another).



GRAPH EXAMPLES

CS prerequisites!
“nodes” are classes
and an “edge” from
class A to class B
means “class B
depends on class A”



GRAPH EXAMPLES

- **To an algorithms person,** a graph means a representation of the relationships between pairs of Objects.
- **Road networks.** When your smartphone's software computes driving directions, it searches through a graph that represents the road network, with vertices corresponding to intersections and edges corresponding to individual road segments.

GRAPH EXAMPLES

- **The World Wide Web.** The Web can be modeled as a directed graph, with the vertices corresponding to individual Web pages, and the edges corresponding to hyperlinks, directed from the page containing the hyperlink to the destination page.
- **Precedence constraints.** Graphs are also useful in problems that lack an obvious network structure. For example, imagine that you a first-year university student, planning which courses to take and in which order.

GRAPH EXAMPLES

- **Social networks.** A social network can be **represented as a graph** whose **vertices** correspond to **individuals** and **edges** to **some type of relationship**. For example, an edge could indicate a friendship between its endpoints, or that one of its endpoints is a follower of the other. Among the currently popular social networks, which ones are most naturally modeled as an **undirected** graph, and which ones as a **directed** graph?

WHAT ARE GRAPHS USED FOR?

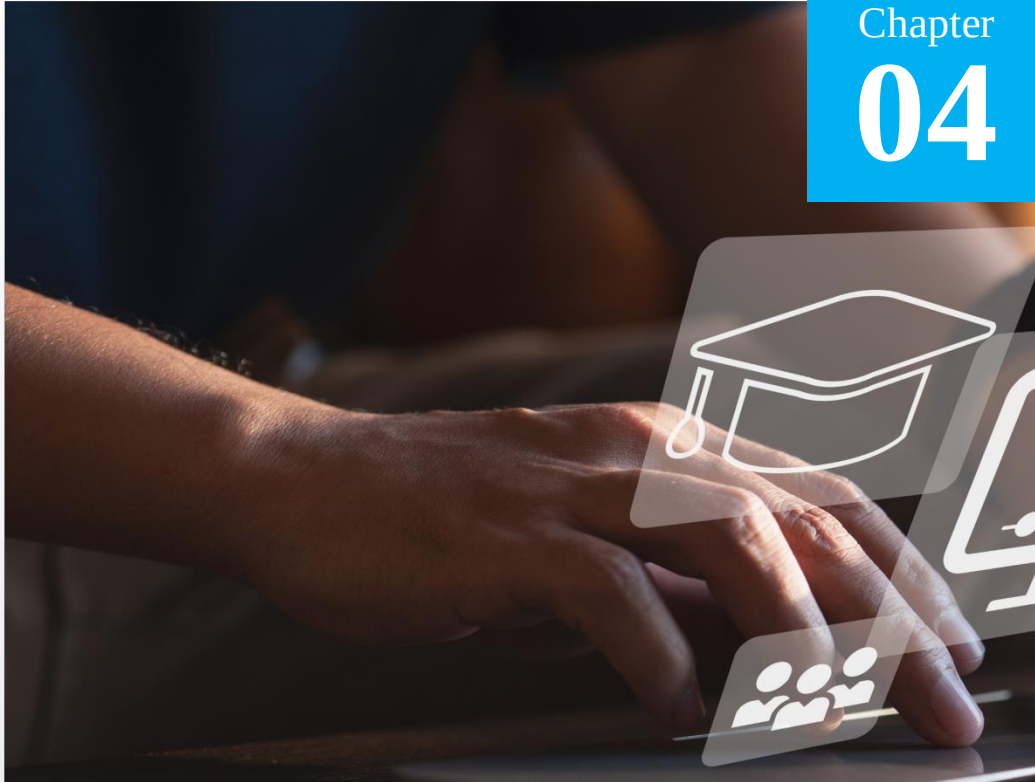
- There are a lot of diverse problems that can be represented as graphs, and we want to answer questions about them
- For example:
 - How do we most efficiently route packets across the internet?
 - Are there natural “clusters” or “communities” in a graph?
 - Which character(s) are least related with _____?
 - How should I sign up for classes without violating pre-req constraints?

But first off, some terminology!

Chapter

04

KINDS OF GRAPHS



TWO KINDS OF GRAPHS:

1. Undirected Graph
2. Directed Graph

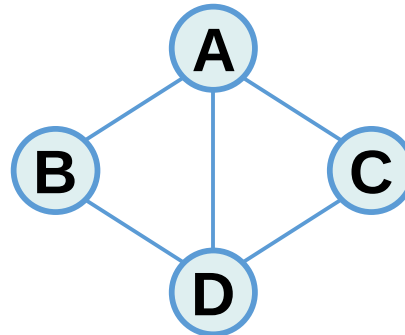
Two KINDS OF GRAPHS:

- In an **undirected graph**, each edge corresponds to an **unordered pair** $\{v, w\}$ of vertices, which are called the endpoints of the edge. In an undirected graph, **there is no difference between an edge (v, w) and an edge (w, v) .**

UNDIRECTED GRAPHS

An undirected graph has
a set of vertices (V) & a set of edges (E)

Formally,
 $G = (V, E)$



$V = \{A, B, C, D\}$

$E = \{ \{A, B\}, \{A, C\}, \{A, D\}, \{B, D\}, \{C, D\} \}$

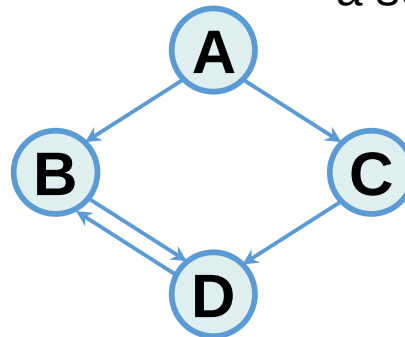
Two KINDS OF GRAPHS:

- In a **directed graph**, each edge (v,w) is an **ordered** pair, with the edge traveling from the first vertex v (called the **tail**) to the second w (the **head**);

DIRECTED GRAPHS

Formally,
 $G = (V, E)$

A directed graph has
a set of vertices (V) & a set of
DIRECTED edges (E)



The **in-degree** of vertex D is 2. The **out-degree** of vertex D is 1.

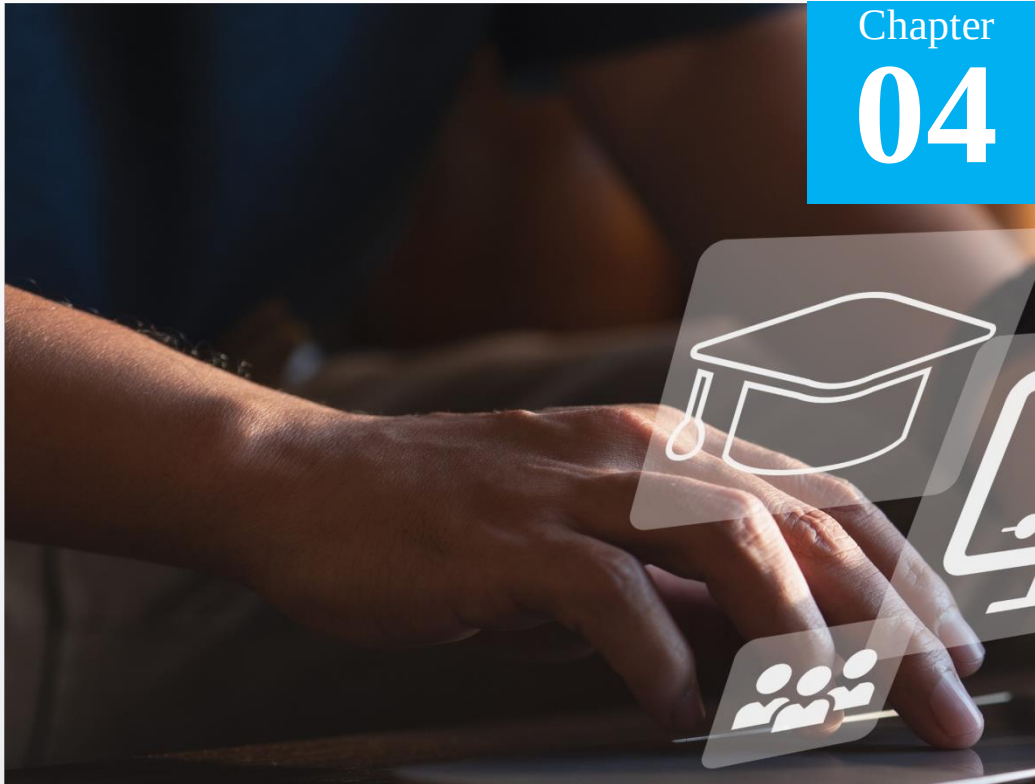
Vertex D's **incoming neighbors** are A, B, & C

Vertex D's **outgoing neighbor** is B

Chapter

04

GRAPH REPRESENTATIONS OPTIONS



GRAPH REPRESENTATIONS OPTIONS:

OPTION 1: ADJACENCY MATRIX

OPTION 2: ADJACENCY LIST

GRAPH REPRESENTATIONS

OPTION 1: ADJACENCY MATRIX

- The adjacency matrix representation of G is a square $n \times n$ matrix A equivalently, a two-dimensional array—with only zeroes and ones as entries. Each entry A_{ij} is defined as:

$$A_{ij} = \begin{cases} 1 & \text{if edge } (i, j) \text{ belongs to } E \\ 0 & \text{otherwise.} \end{cases}$$

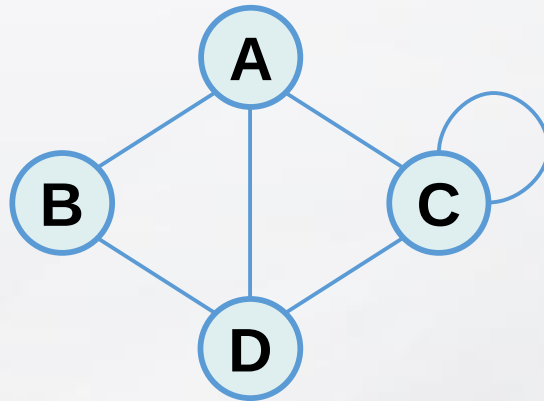
GRAPH REPRESENTATIONS

OPTION 2: ADJACENCY LIST

- Ingredients for Adjacency Lists
 1. An array containing the graph's vertices.
 2. An array containing the graph's edges.
 3. For each edge, a pointer to each of its two endpoints.
 4. For each vertex, a pointer to each of the incident edges.

GRAPH REPRESENTATIONS

OPTION 1: ADJACENCY MATRIX

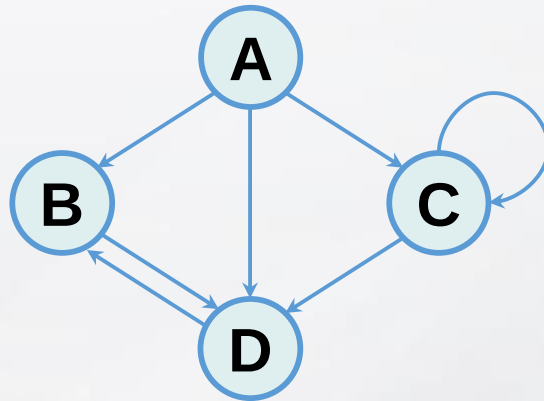


(An undirected graph)

		(destination)			
		A	B	C	D
(source)	A	0	1	1	1
	B	1	0	0	1
	C	1	0	1	1
	D	1	1	1	0

GRAPH REPRESENTATIONS

OPTION 1: ADJACENCY MATRIX

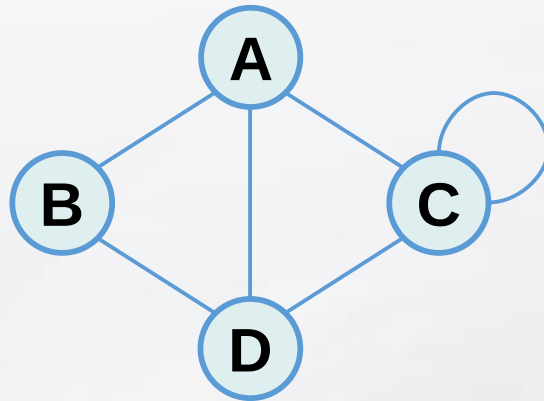


(A directed graph)

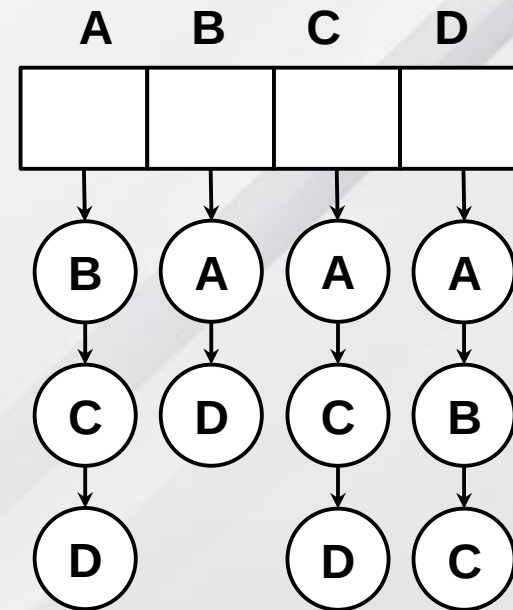
		(destination)			
		A	B	C	D
(source)	A	0	1	1	1
	B	0	0	0	1
	C	0	0	1	1
	D	0	1	0	0

GRAPH REPRESENTATIONS

OPTION 2: ADJACENCY LISTS



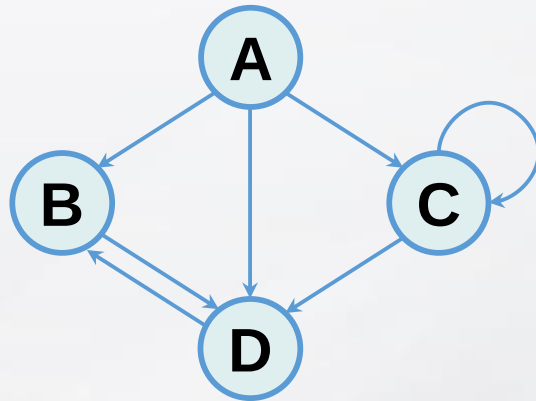
(An undirected graph)



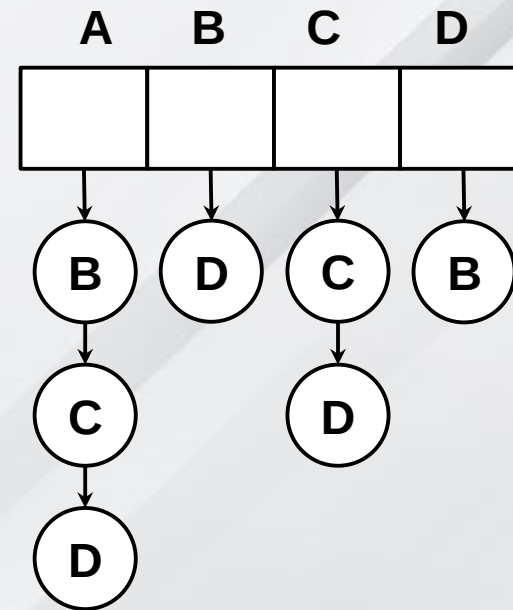
Each list stores a node's neighbors

GRAPH REPRESENTATIONS

OPTION 2: ADJACENCY LISTS



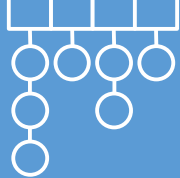
(A directed graph)



Tracks outgoing neighbors.

(You could also do the same for incoming neighbors as well)

GRAPH REPRESENTATIONS

GRAPH REPRESENTATIONS TECHNIQUES	ADJACENCY MATRIX	ADJACENCY LISTS
For a graph $G = (V, E)$ where $ V = n$, and $ E = m$	$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$	
SPACE REQUIREMENTS	$O(n^2)$	$O(n + m)$

Generally, better for sparse graphs (where $m \ll n^2$).

We'll assume this representation, unless otherwise stated.

GRAPH REPRESENTATIONS

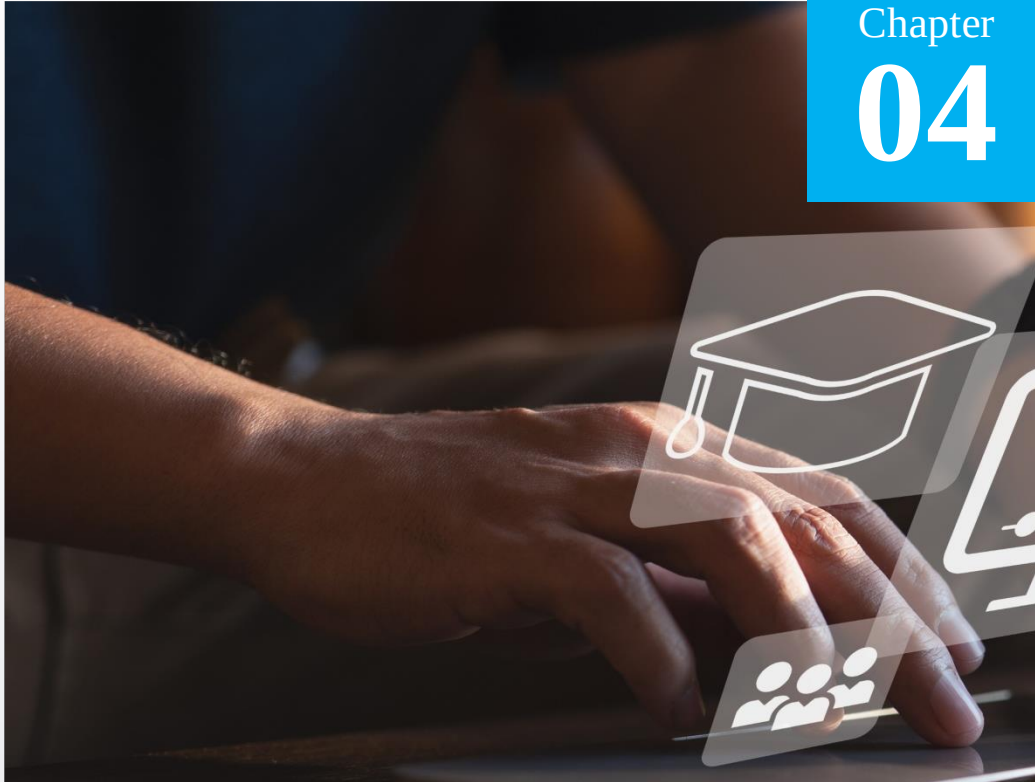
Generally Adjacency Lists representation, better than Adjacency Matrix representation for sparse graphs (where $m < n^2$).

We'll assume Adjacency Lists representation, unless otherwise stated.

Chapter

04

Sparse Graphs vs. Dense Graphs



Sparse Graphs vs. Dense Graphs

- A graph is **sparse** if the number of edges is relatively close to linear in the **number of vertices**.
- A graph is **dense** if the number of edges is closer to quadratic in the **number of vertices**.

BREADTH-FIRST SEARCH (BFS)

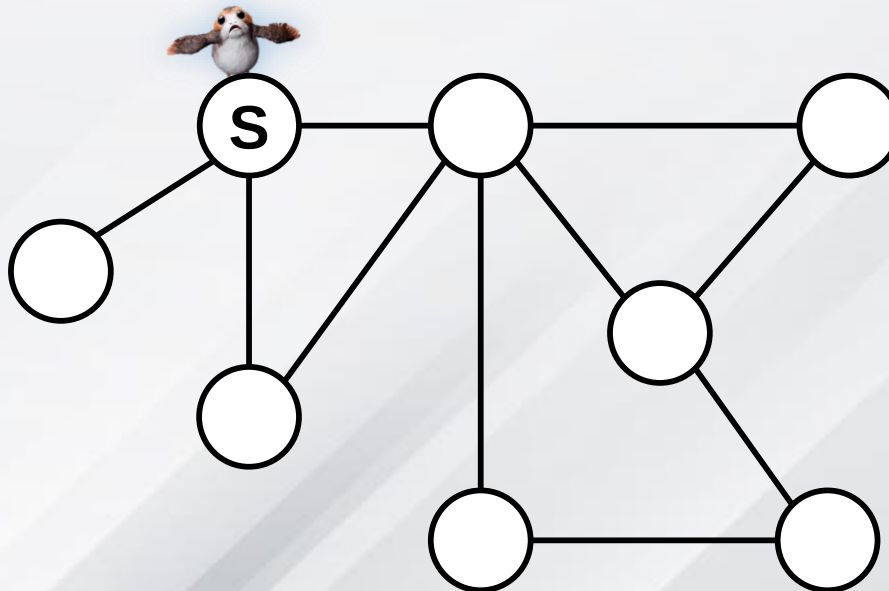
One way to explore a graph!

BREADTH-FIRST SEARCH

- Breadth-first search explores the vertices of a graph in layers, in order of increasing distance from the starting vertex.

An analogy:

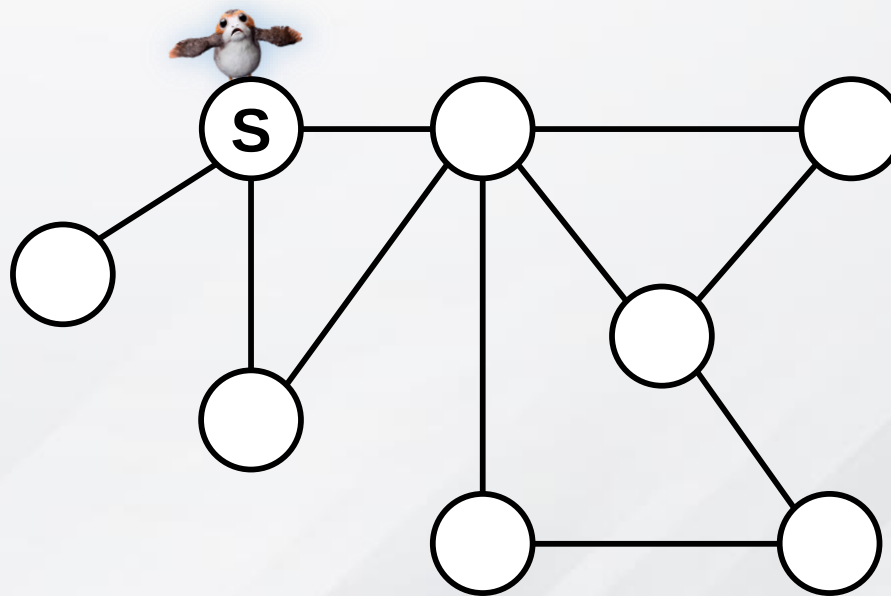
A bird is exploring a maze from above (with a bird's eye view)

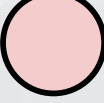


BREADTH-FIRST SEARCH

An analogy:

A bird is exploring a labyrinth from above (with a bird's eye view)

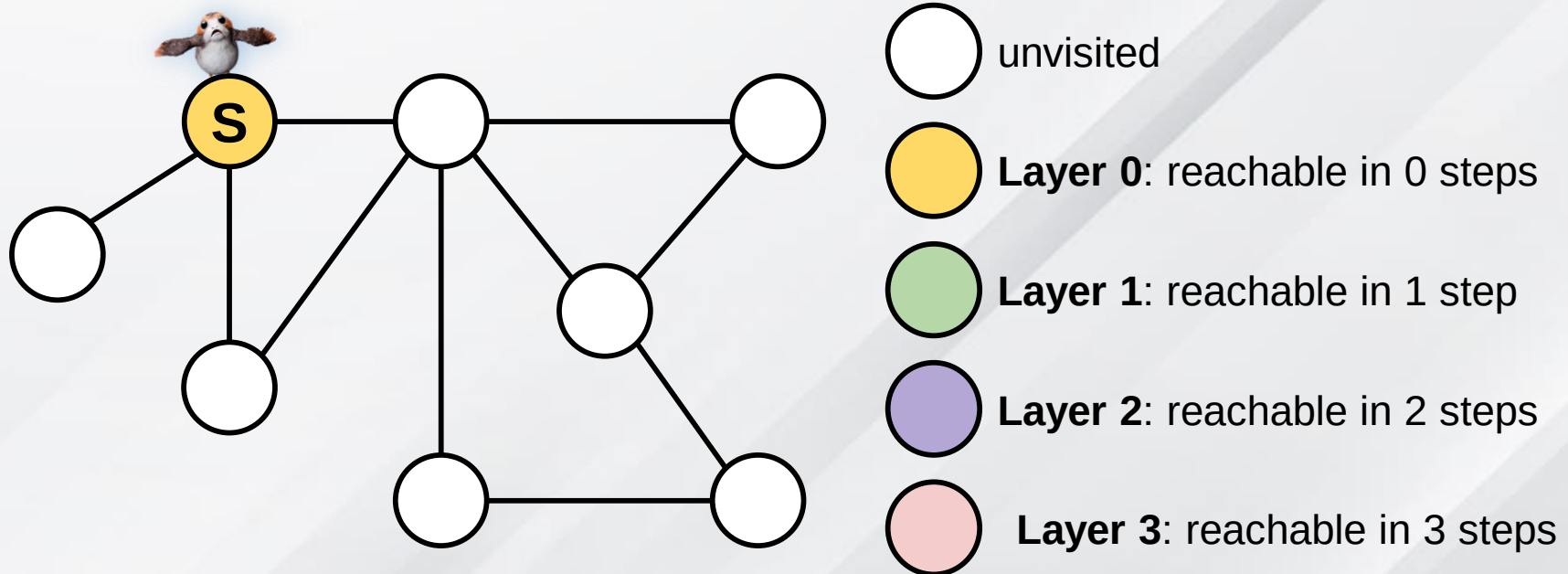


-  unvisited
-  **Layer 0:** reachable in 0 steps
-  **Layer 1:** reachable in 1 step
-  **Layer 2:** reachable in 2 steps
-  **Layer 3:** reachable in 3 steps

BREADTH-FIRST SEARCH

An analogy:

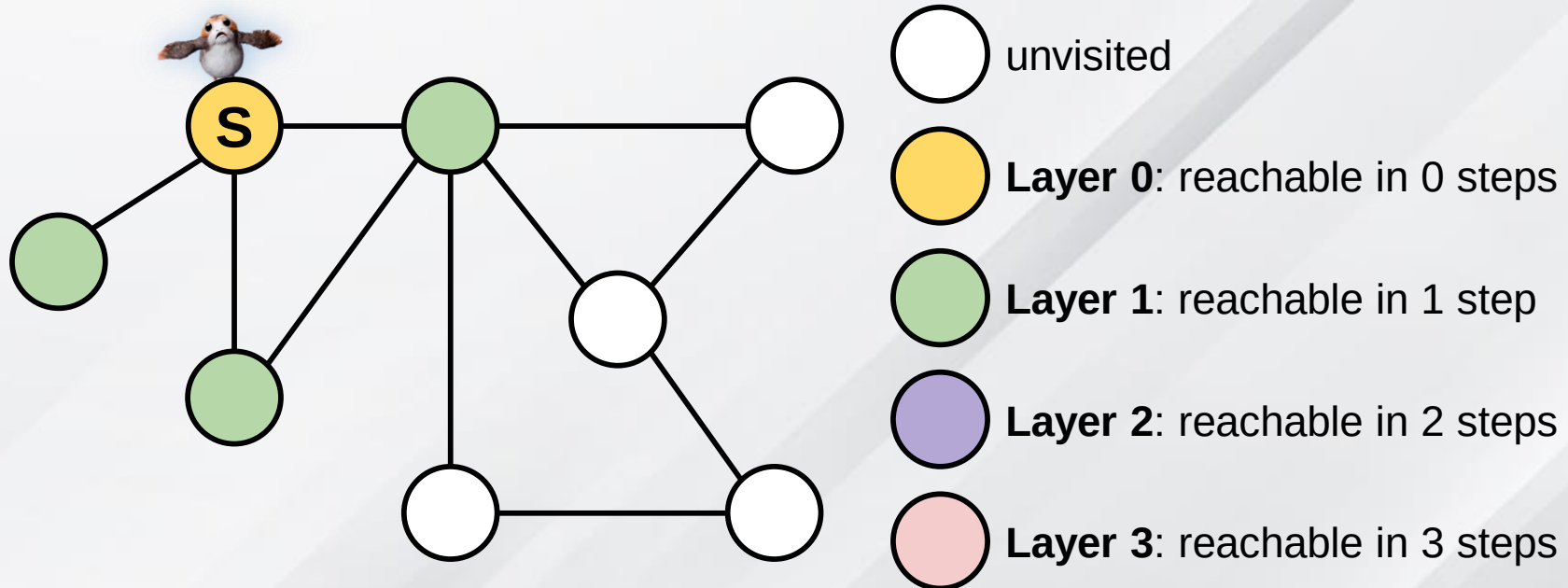
A bird is exploring a labyrinth from above (with a bird's eye view)



BREADTH-FIRST SEARCH

An analogy:

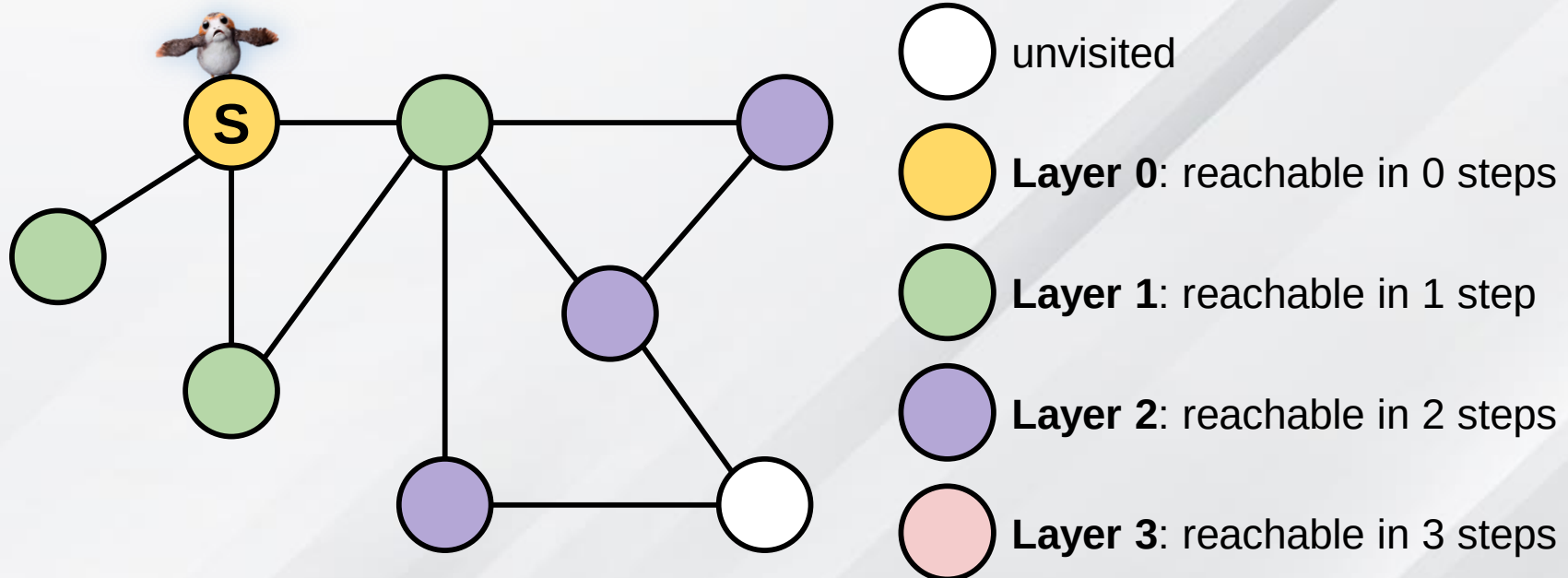
A bird is exploring a labyrinth from above (with a bird's eye view)



BREADTH-FIRST SEARCH

An analogy:

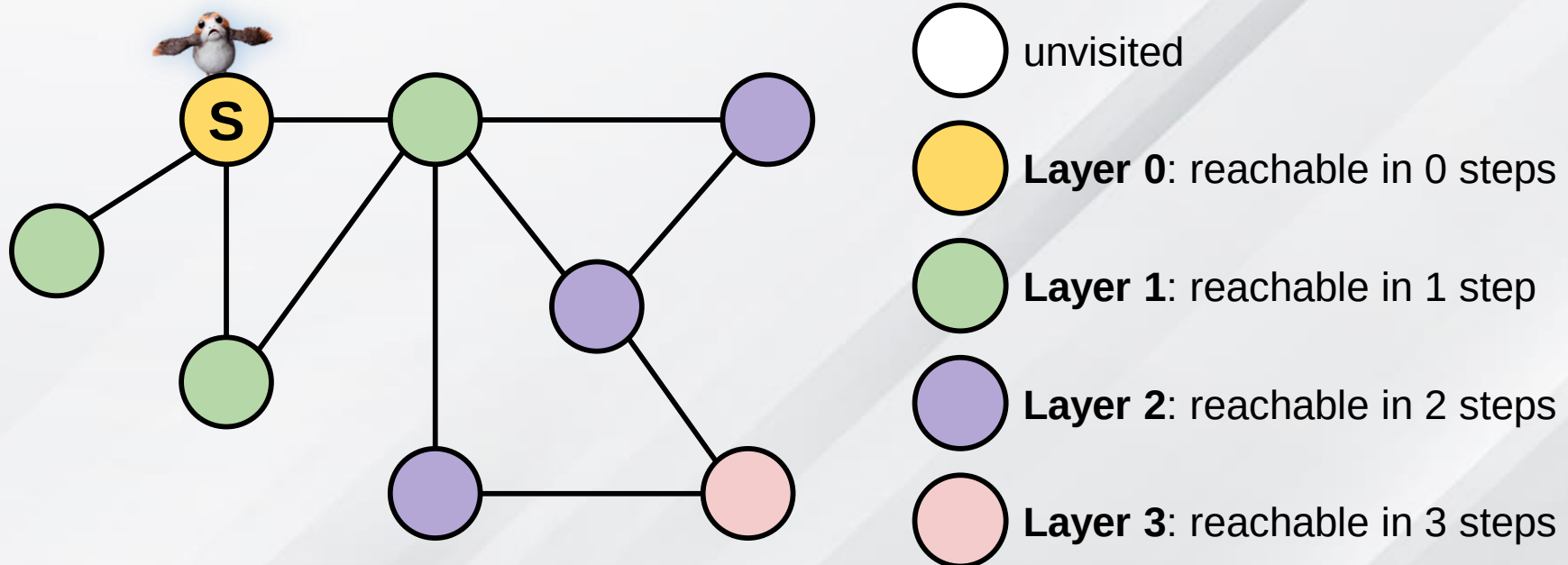
A bird is exploring a labyrinth from above (with a bird's eye view)



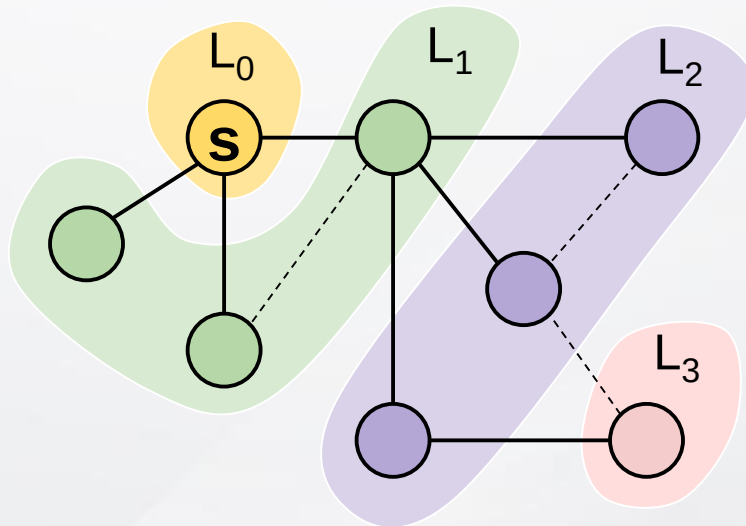
BREADTH-FIRST SEARCH

An analogy:

A bird is exploring a labyrinth from above (with a bird's eye view)



BREADTH-FIRST SEARCH



L_i = The set of nodes we can reach in i steps from s

BFS(s):

Set $L_i = []$ for $i = 0, \dots, n-1$

$L_0 = s$

for $i = 0, \dots, n-1$:

for u in L_i :

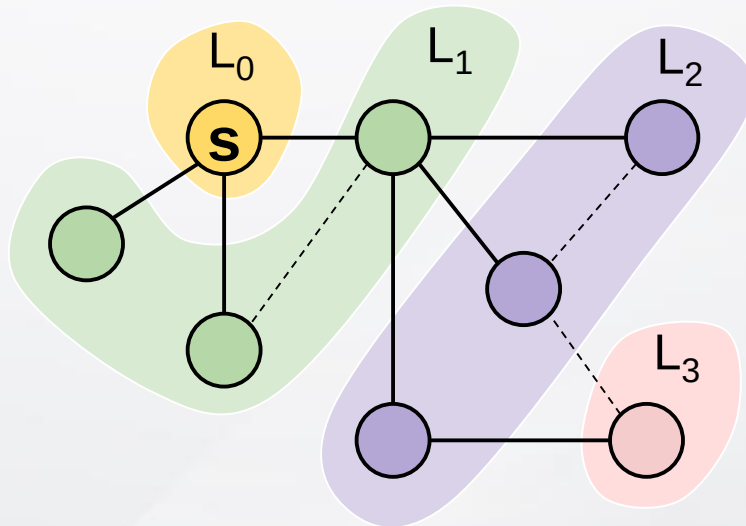
for v in u .neighbors:

if v not yet visited:

mark v as visited

add v to L_{i+1}

BREADTH-FIRST SEARCH



L_i = The set of nodes we can reach in i steps from s

BFS(s):

Set $L_i = []$ for $i = 0, \dots, n-1$

$L_0 = s$

for $i = 0, \dots, n-1$:

for u in L_i :

for v in u .neighbors:

if v not yet visited:

mark v as visited

add v to L_{i+1}

Go through all nodes in L_i and add their unvisited neighbors to L_{i+1}



Thank You