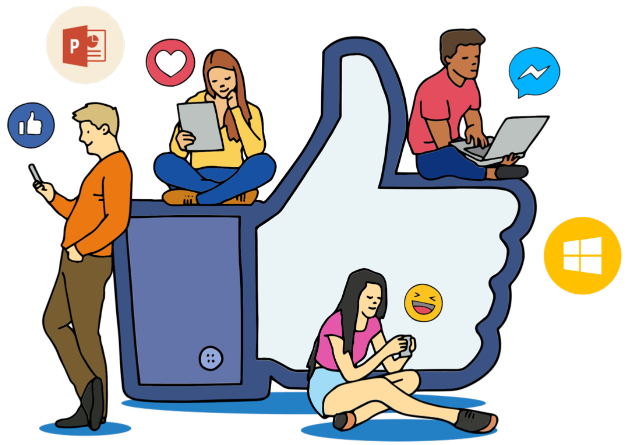
INFS7450 SOCIAL MEDIA ANALYTICS
Week 2

Liang QU

School of EECS

The University of Queensland



About me: Liang QU

- Office room: 78-622
- Email: liang.qu@uq.edu.au
- Face-to-face: appointment by email
- Research interests: graph embedding, recommender systems, LLM

Today's Contents

- About INFS7450
- Graph Essentials
- Coding Demo
- Q&A

Section 1: Recall Key Points on INFS7450

Section 1: Course Schedule

	Topic	Lecturer
Week1-Wednesday	Course Introduction	Dr. Hongzhi Yin
Week2-Wednesday	Review of Graph Essentials	Dr. Hongzhi Yin
Week3-Wednesday	Introduction to Node Measures	Dr. Hongzhi Yin
Week4-Wednesday	Computation of Node Centrality Measures	Dr. Hongzhi Yin
Week5-Wednesday	Network Measures and Models	Dr. Hongzhi Yin
Week6-Wednesday	Influence and Homophily	Dr. Hongzhi Yin
	In-Semester Break	
Week7-Wednesday	Network Effects and Cascading Behaviour	Dr. Hongzhi Yin
Week8-Wednesday	Community Structure and Detection in Networks	Dr. Hongzhi Yin
Week9-Wednesday	Link Prediction	Dr. Hongzhi Yin
Week10-Wednesday	Traditional Machine Learning with Graphs	Dr. Hongzhi Yin
Week11-Wednesday	Graph Representation Learning I	Dr. Junliang Yu
Week12-Wednesday	Graph Representation Learning II	Dr. Junliang Yu
Week13-Wednesday	Course Review	Dr. Hongzhi Yin

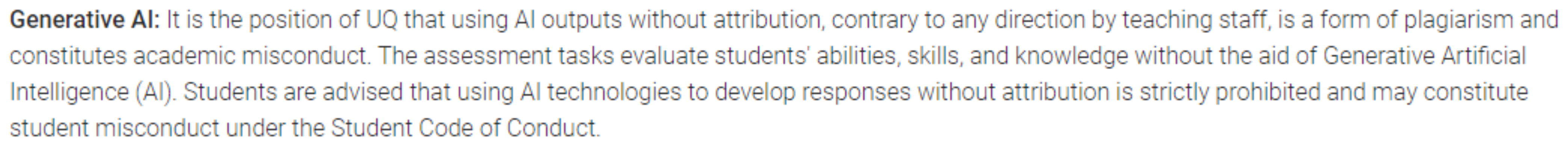
Assignments	Release Date (Brisbane Time)	Due Date (Brisbane Time)
Online Quiz 1	Wednesday, 28/02/2024	06/03/2024-16:00
Online Quiz 2	Wednesday, 06/03/2024	20/03/2024-16:00
Online Quiz 3	Wednesday, 20/03/2024	27/03/2024-16:00
Online Quiz 4	Wednesday, 27/03/2024	10/04/2024-16:00
Project 1	Wednesday, 13/03/2024	17/04/2024-16:00
Project 2	Wednesday, 24/04/2024	22/05/2024-16:00

The 1st online quiz has been already released!
Do not forget it!

Section 1: Course Website - Blackboard

- Lectures, Tutorials and Their Recordings
  - Slides posted at the night before the lecture and tutorial sessions respectively.
  - Their recordings will be available on Blackboard system.
- Assignments
  - Both online quizzes and projects will be released and submitted via Blackboard system.
- Communication
  - All announcements and reminders about the deadline and due date will be posted through Blackboard.
  - **Ed Discussion Board** will be used for discussions, queries and questions.
    - Emails to the tutors for private questions; all other questions should be posted to ED.
    - Students are encouraged to help each other on the discussion board.
    - If you cannot access this course on Ed Discussion, please contact Skye(skye.sun@uq.edu.au)

Section 1: About Generative AI


Generative AI: It is the position of UQ that using AI outputs without attribution, contrary to any direction by teaching staff, is a form of plagiarism and constitutes academic misconduct. The assessment tasks evaluate students' abilities, skills, and knowledge without the aid of Generative Artificial Intelligence (AI). Students are advised that using AI technologies to develop responses without attribution is strictly prohibited and may constitute student misconduct under the Student Code of Conduct.

Examples of misuse:

- Reports: Submitting a report generated by AI without indicating that AI was used.
- Code: Incorporating AI-generated code in your projects without citing the AI as a source.
- Quizzes: Using AI to answer quiz questions

Failure to properly cite AI contributions is a form of academic misconduct
Always attribute AI assistance in your work

Course Profiles: https://course-profiles.uq.edu.au/student_section_loader/section_1/132633

Section 2: Graph Essentials

Section 2: In this section, we'd like to answer:

- Types of graphs
  - Directed graph VS undirected graph
  - Weighted graph VS unweighted graph
  - Regular graph VS complete graph
  - Simple graph VS multigraph
- How to traverse a graph?
  - DFS
  - BFS
- What is a strongly connected component in a directed graph?
- How to find the shortest path in a weighted graph?

Section 2: Types of Graphs

- Directed graph VS undirected graph
  - Handshake Theorem: Let G=(V,E) be an undirected graph with m edges. Then

$$2m = \sum_{v \in V} \deg(v)$$

Proof: Each edge contributes twice to the total degree count of all vertices. Thus, both sides of the equation equal to twice the number of edges.

  - Theorem: An undirected graph has an even number of nodes of odd degree.

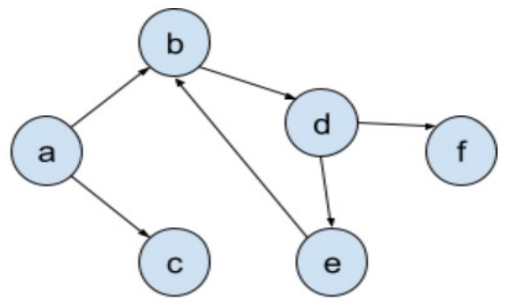
Proof: Let V_1 be the nodes of even degree and V_2 be the nodes of odd degree in an undirected graph G=(V,E) with m edges. Then

$$2m = \sum_{v \in V} \deg(v) = \sum_{v \in V_1} \deg(v) + \sum_{v \in V_2} \deg(v)$$

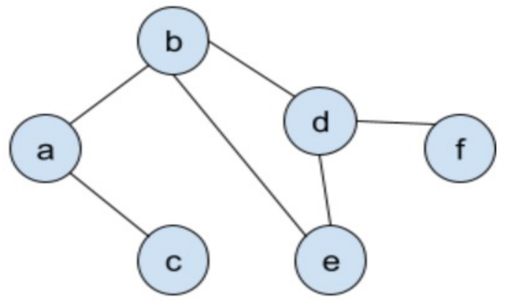
Must be even since deg(v) is even for each $v \in V_1$

This sum must be even because 2m is even and the sum of the degrees of the nodes of even degrees is also even. Because this is the sum of the degrees of all nodes of odd degree in the graph, there must be an even number of such nodes.

Directed
Links: directed



Undirected
Links: undirected (symmetrical, reciprocal)

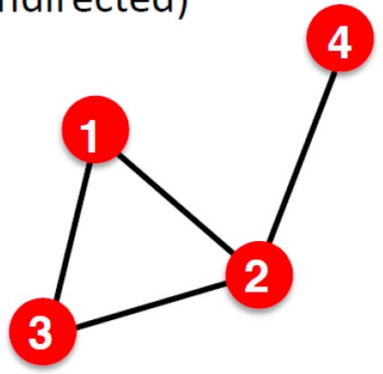


Section 2: Types of Graphs

- Weighted graph VS unweighted graph
  - Degree has generally been extended to the sum of weights when analyzing weighted graph.

▪ **Unweighted**

(undirected)

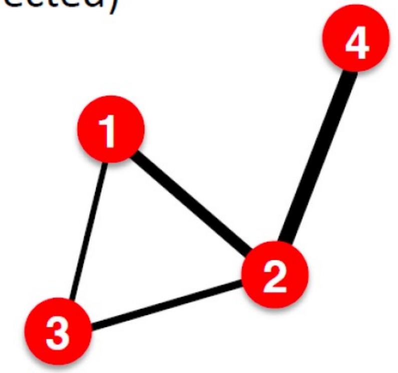


$$A_{ij} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$A_{ii} = 0 \qquad A_{ij} = A_{ji}$

▪ **Weighted**

(undirected)

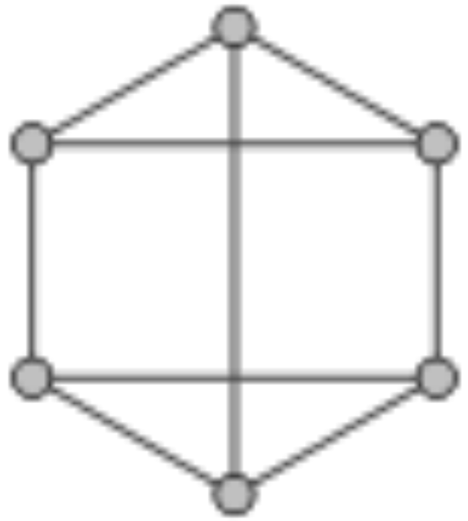


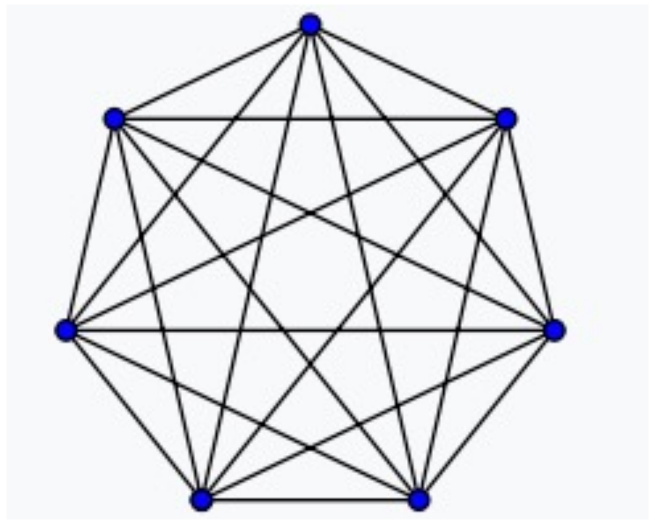
$$A_{ij} = \begin{pmatrix} 0 & 2 & 0.5 & 0 \\ 2 & 0 & 1 & 4 \\ 0.5 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 \end{pmatrix}$$

$A_{ii} = 0 \qquad A_{ij} = A_{ji}$

Section 2: Types of Graphs

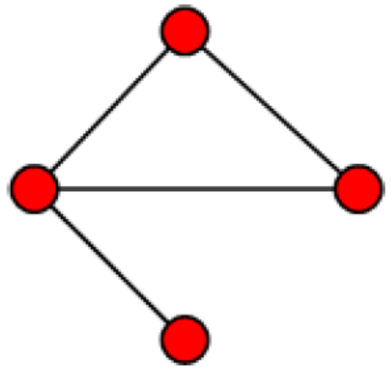
- Regular graph VS complete graph
  - A regular graph is a graph where each node has the same number of neighbors; i.e., every node has the same degree.
  - A complete graph is a graph in which each pair of graph nodes is connected by an edge.
  - A complete graph is always a regular graph.

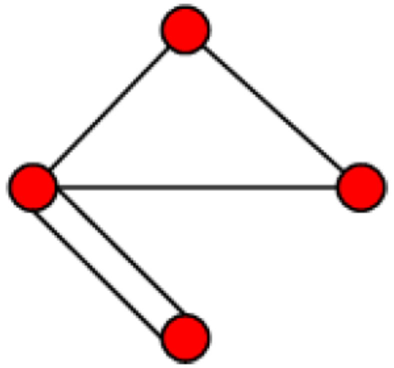

3-regular graph


Complete graph

Section 2: Types of Graphs

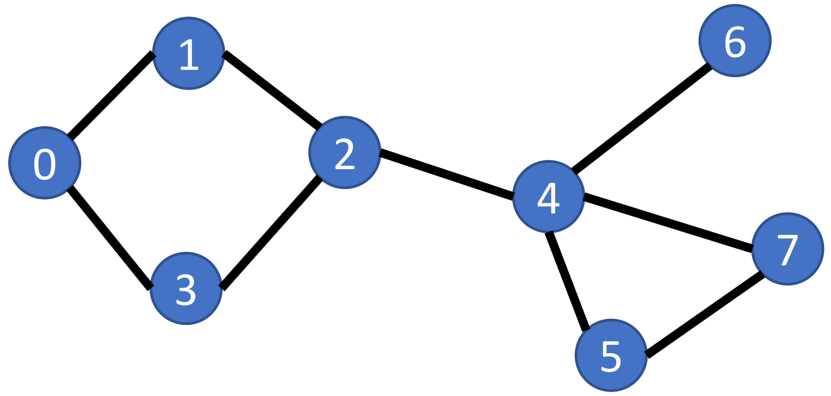
- Simple graph VS multigraph
  - Simple graphs are graphs where only a single edge is allowed to exist between any pair of nodes (contain no self loops)
  - Multigraphs are graphs where you can have multiple edges between two nodes (loops are allowed)


Simple graph


Multigraph

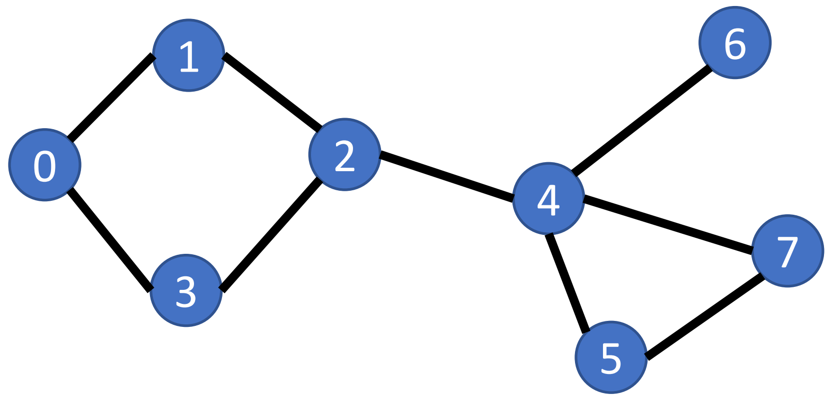
Section 2: How to traverse a graph?

- Graph traversal refers to the process of visiting each node in a graph.
- Typical solution:
  - Depth First Search (DFS)
  - Breath First Search (BFS)



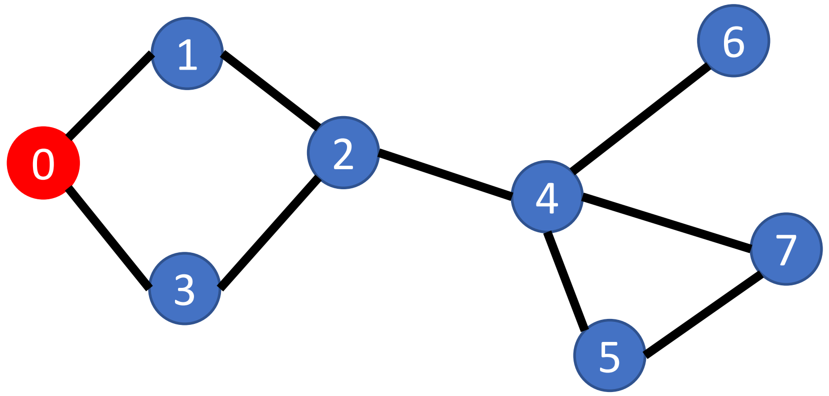
Section 2: DFS

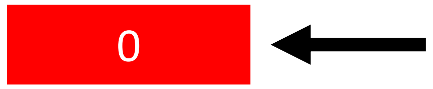
- The Depth First Search (DFS) is the most fundamental search algorithm used to explore nodes and edges of a graph.
- As the name suggests, a DFS plunges depth first into a graph without regard for which edge it takes next until it cannot go any further at which point it backtracks and continues.



Section 2: DFS

- Start DFS at node 0
- The node 0 is in the stack.

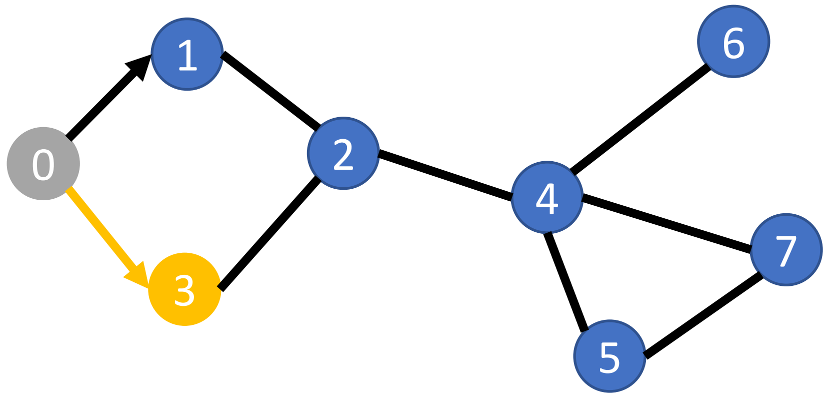


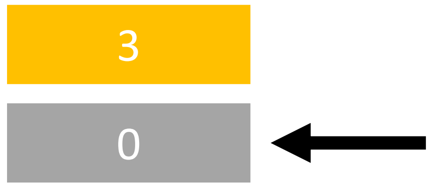


The order of visited nodes: {}

Section 2: DFS

- The node 0 is out of stack
- Put the neighbor nodes of node 0 to the stack

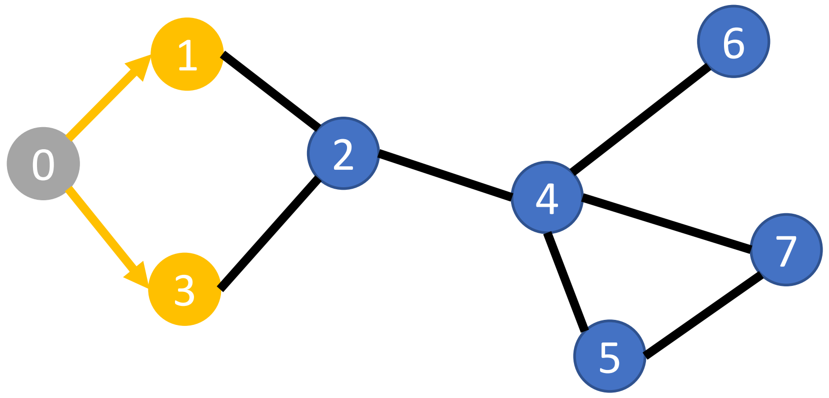


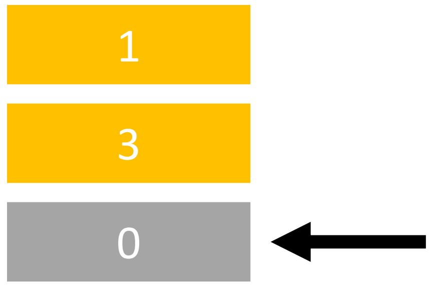


The order of visited nodes: {0}

Section 2: DFS

- The node 0 is out of stack
- Put the neighbor nodes of node 0 to the stack

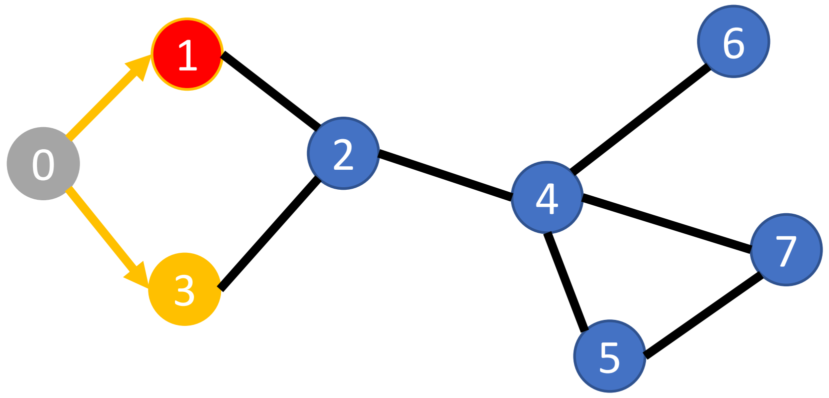


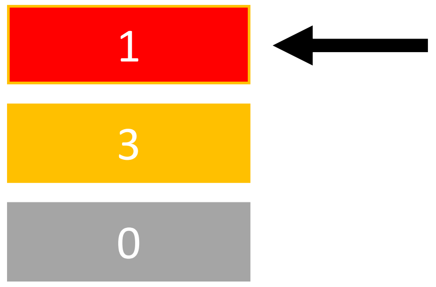


The order of visited nodes: {0}

Section 2: DFS

- Stack follows the LIFO (Last in First out) principle
- Node 1 is going out of the stack

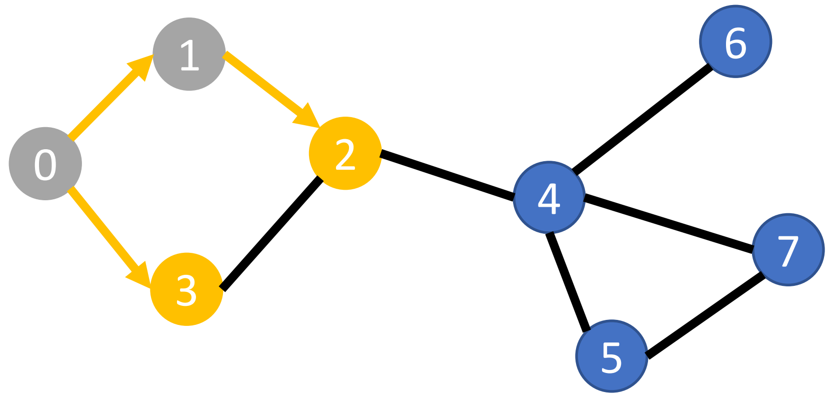


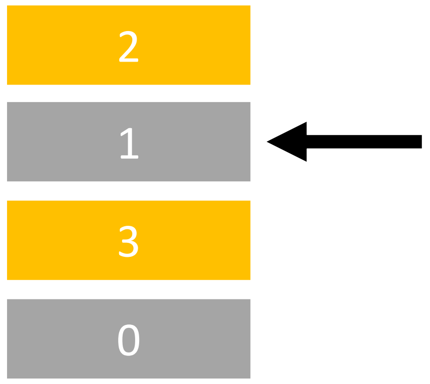


The order of visited nodes: {0}

Section 2: DFS

- Node 1 is out of the stack
- Put the neighbor node of node 1 into the stack

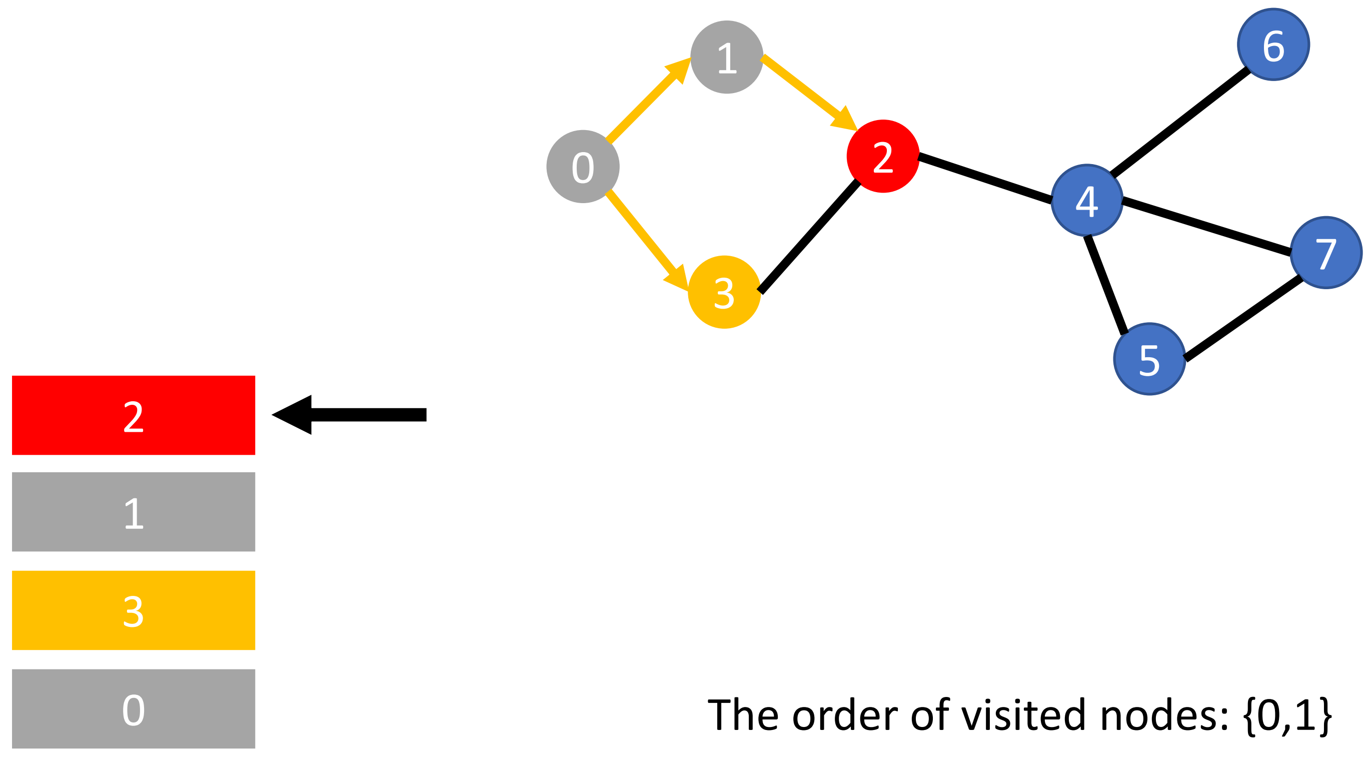




The order of visited nodes: {0,1}

Section 2: DFS

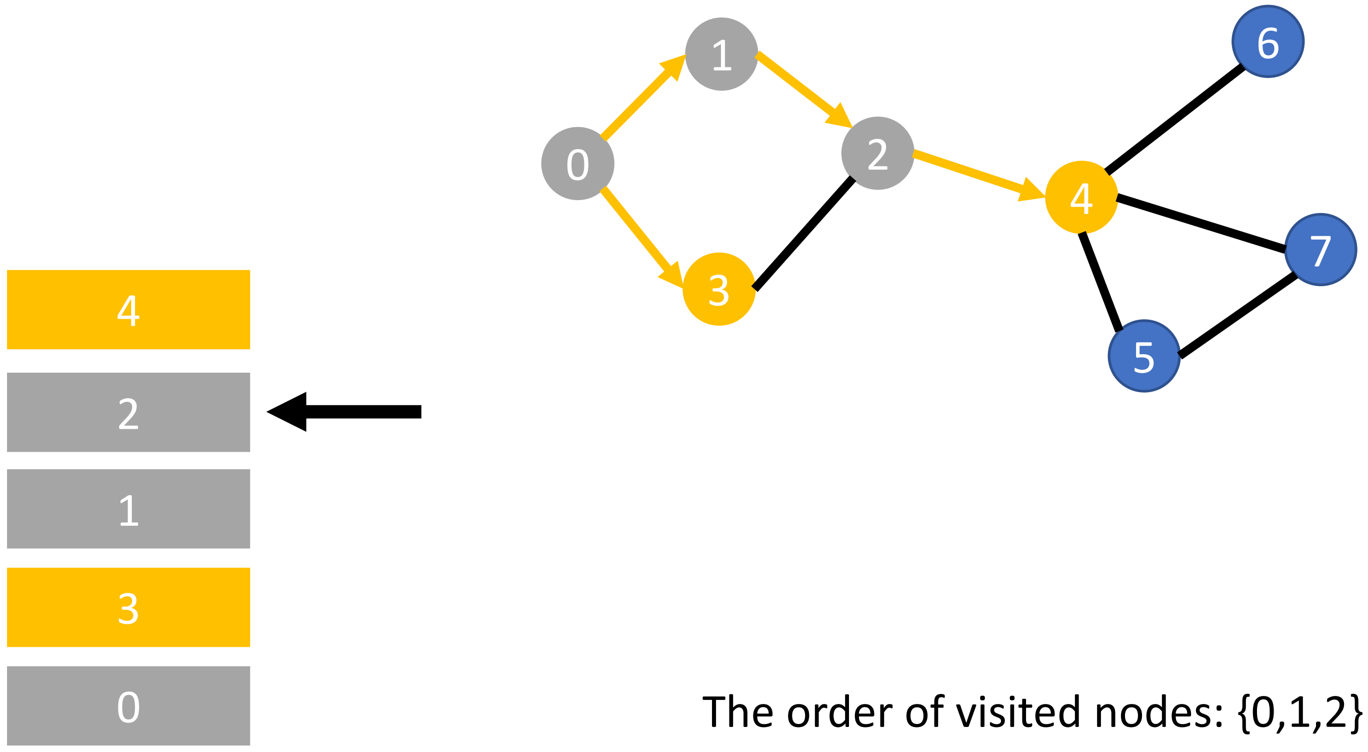
- Stack follows the LIFO (Last in First out) principle
- Node 2 is going out of the stack



The order of visited nodes: {0,1}

Section 2: DFS

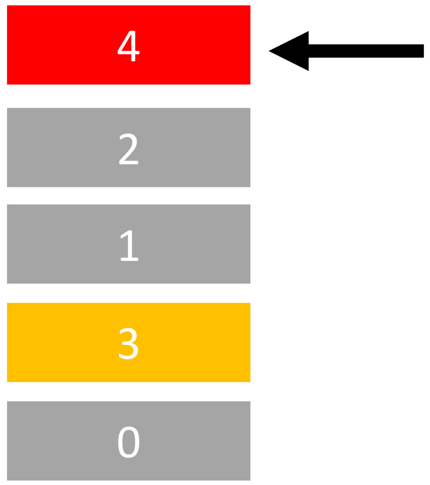
- Node 2 is out of the stack
- Put the neighbor node of node 2 into the stack

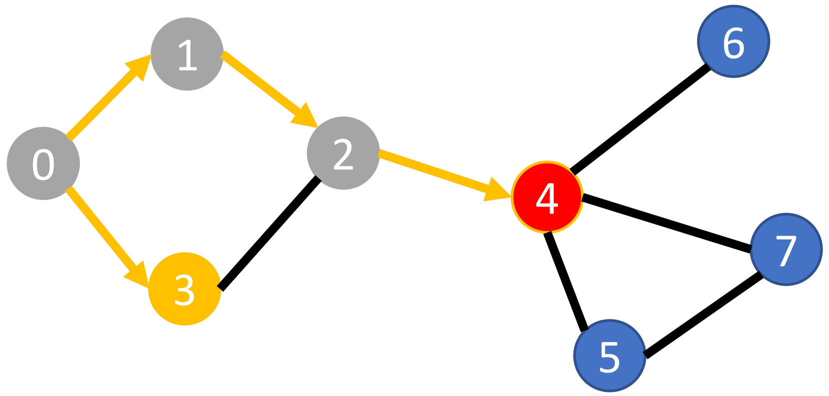


The order of visited nodes: {0,1,2}

Section 2: DFS

- Stack follows the LIFO (Last in First out) principle
- Node 4 is going out of the stack

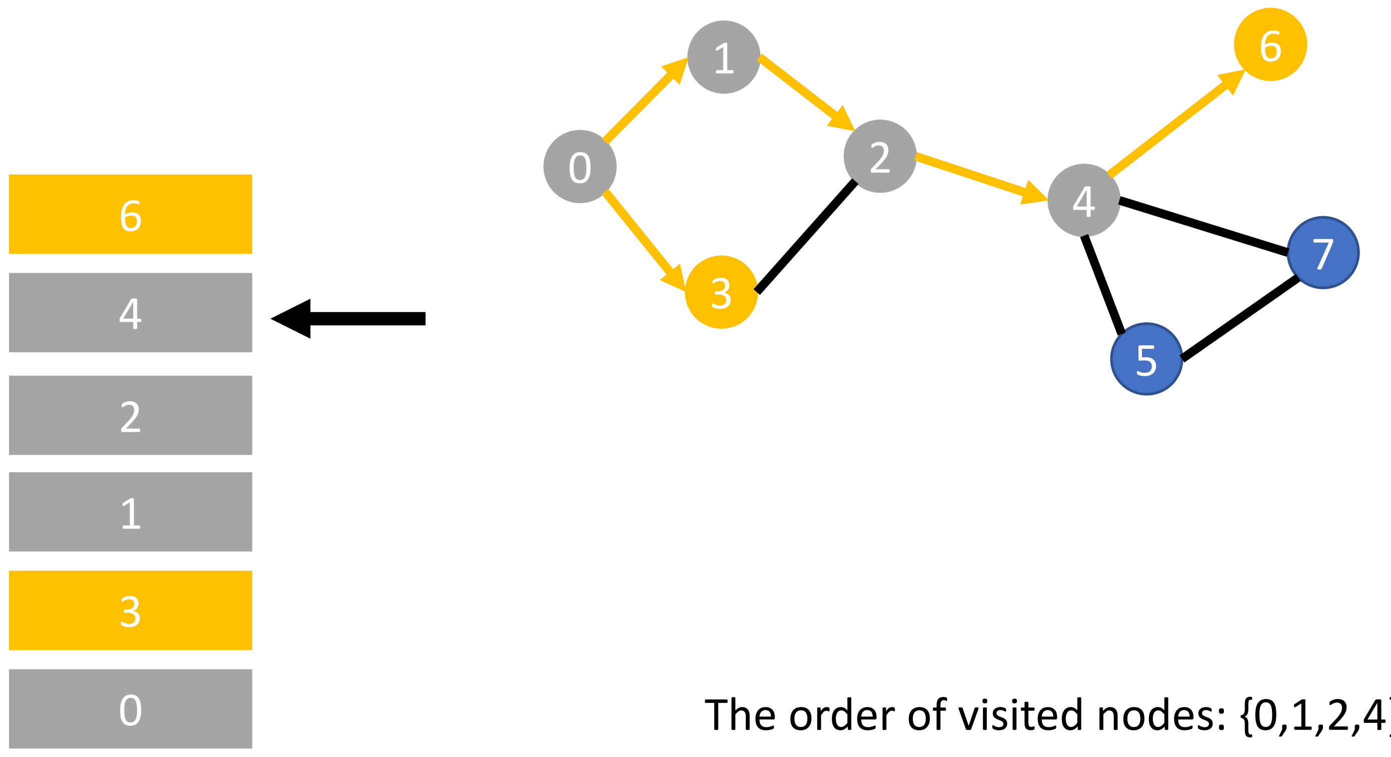




The order of visited nodes: {0,1,2}

Section 2: DFS

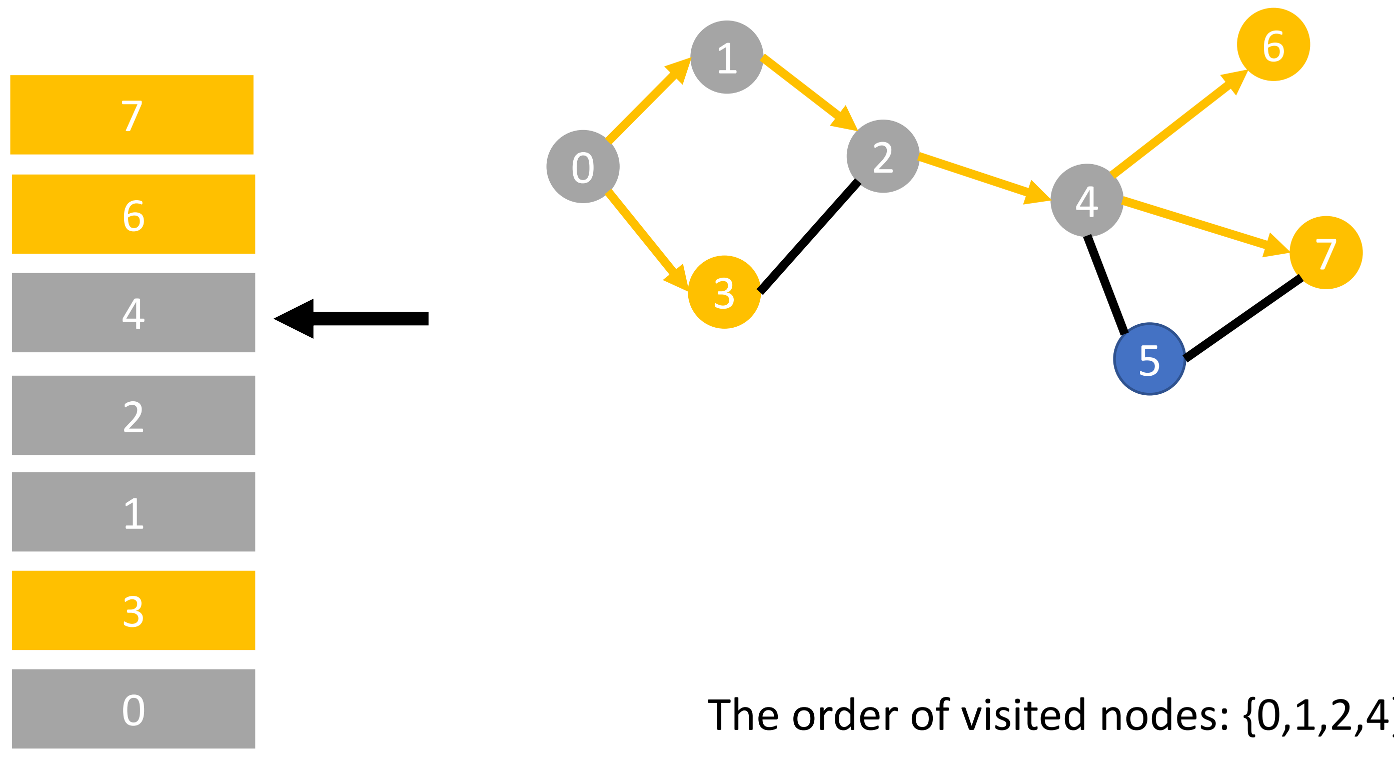
- Node 4 is out of the stack
- Put the neighbor node of node 4 into the stack



The order of visited nodes: {0,1,2,4}

Section 2: DFS

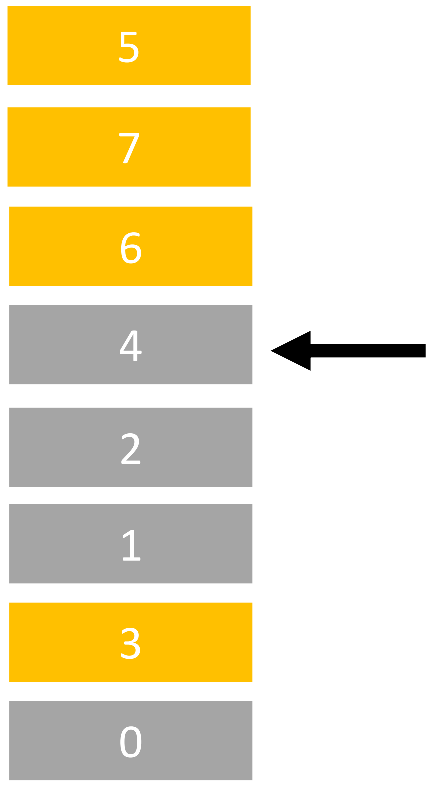
- Node 4 is out of the stack
- Put the neighbor node of node 4 into the stack

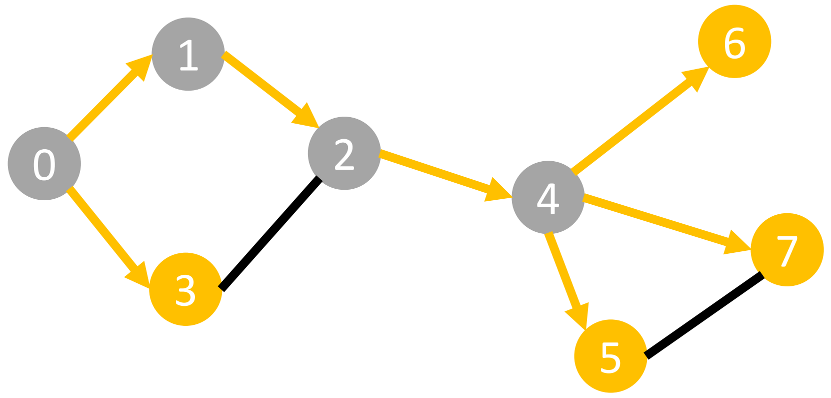


The order of visited nodes: {0,1,2,4}

Section 2: DFS

- Node 4 is out of the stack
- Put the neighbor node of node 4 into the stack

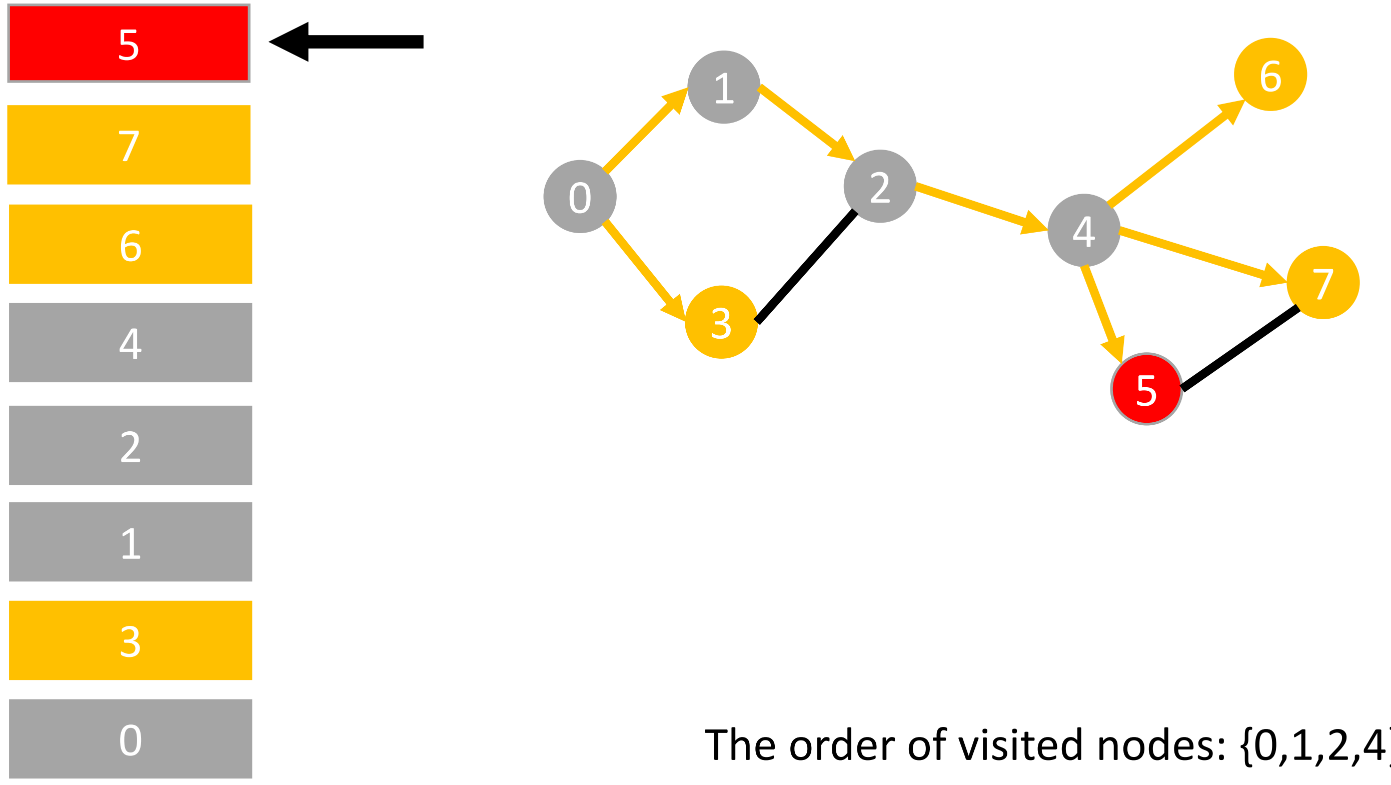




The order of visited nodes: {0,1,2,4}

Section 2: DFS

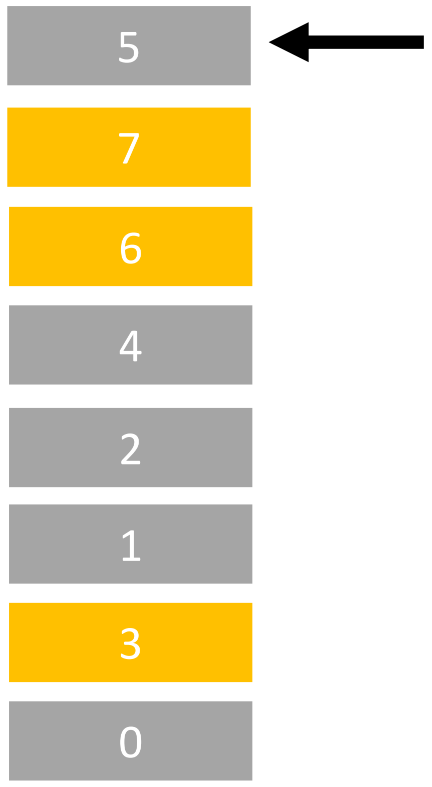
- Stack follows the LIFO (Last in First out) principle
- Node 5 is going out of the stack

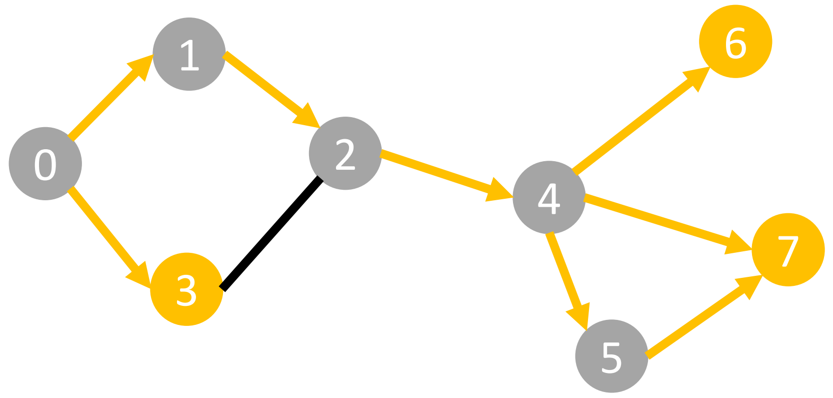


The order of visited nodes: {0,1,2,4}

Section 2: DFS

- Node 5 is out of the stack
- Put the neighbor node of node 5 into the stack
- Node 7 is already in the stack

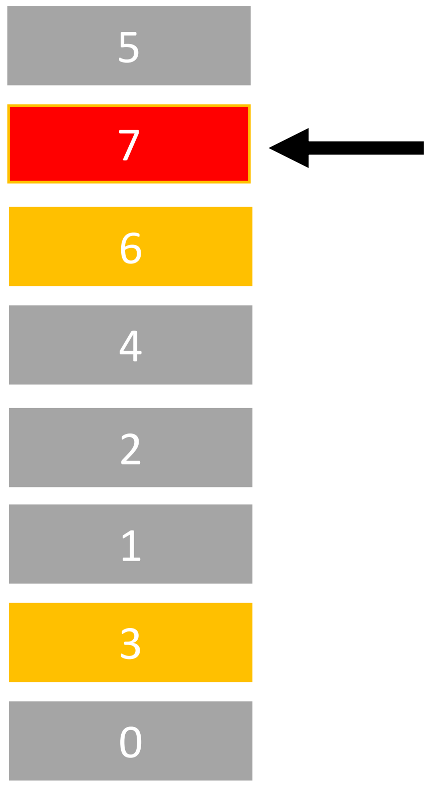


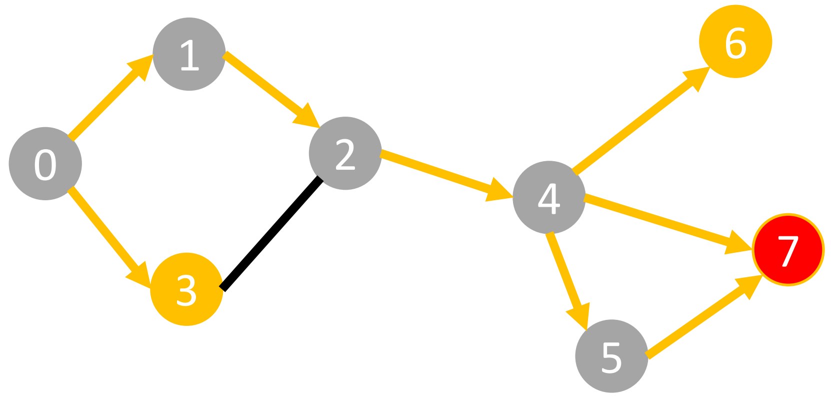


The order of visited nodes: {0,1,2,4,5}

Section 2: DFS

- Stack follows the LIFO (Last in First out) principle
- Node 7 is going out of the stack

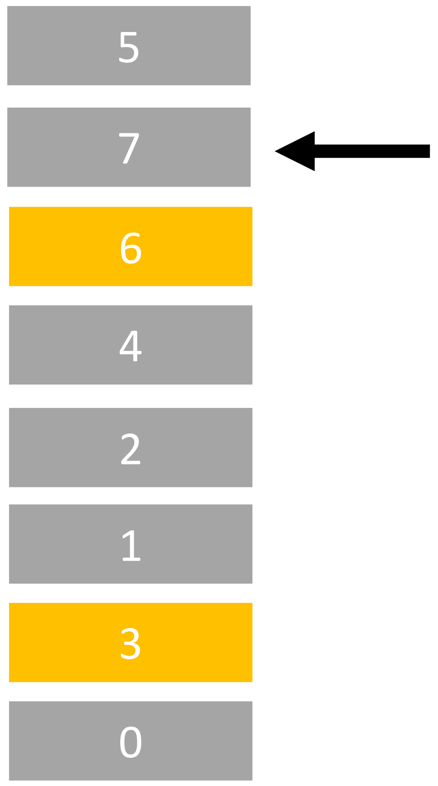


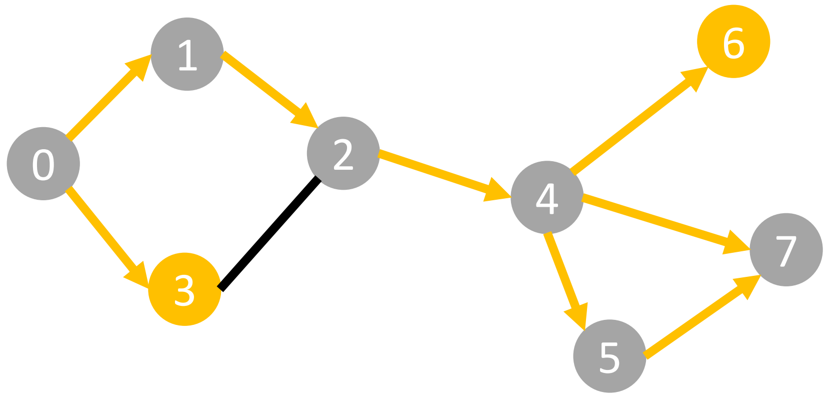


The order of visited nodes: {0,1,2,4,5}

Section 2: DFS

- Node 7 is out of the stack
- Backtrack the stack
- Make sure you don’t re-visit visited nodes!

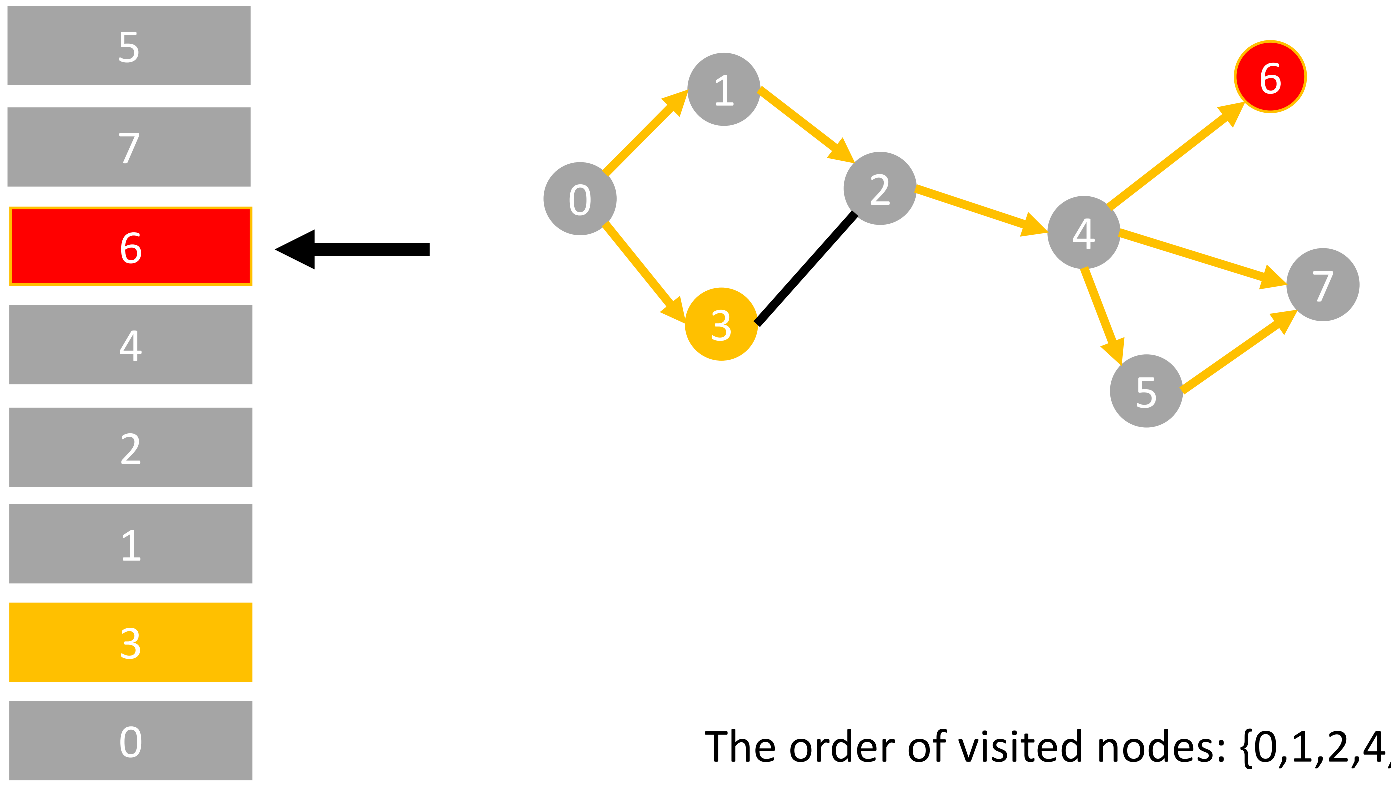




The order of visited nodes: {0,1,2,4,5,7}

Section 2: DFS

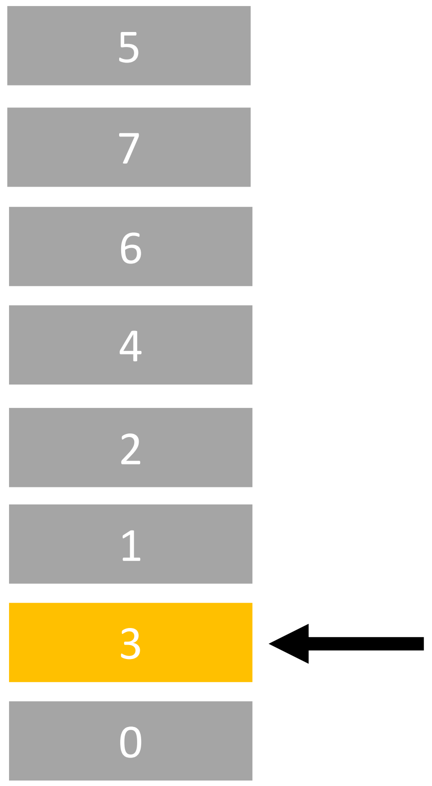
- Node 6 is going out of the stack
- Backtrack when a dead end is reached.

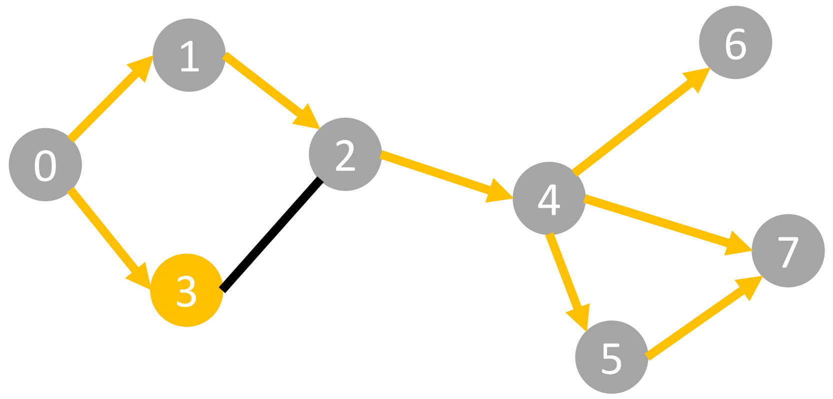


The order of visited nodes: {0,1,2,4,5,7}

Section 2: DFS

- Node 6 is out of the stack
- Backtrack

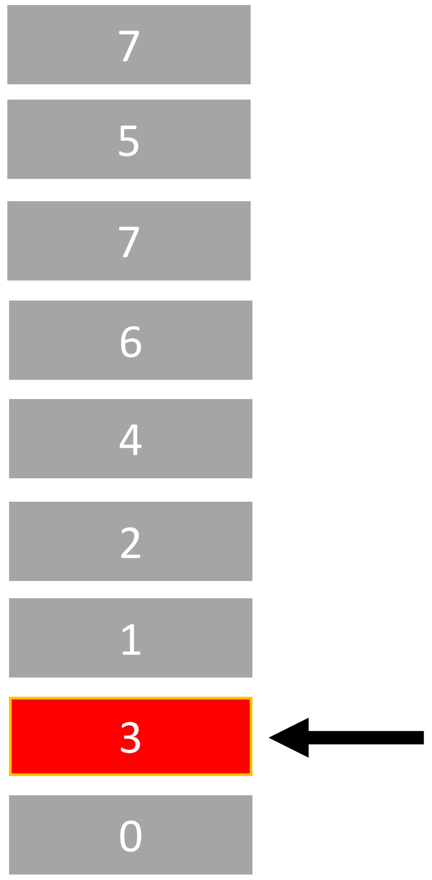


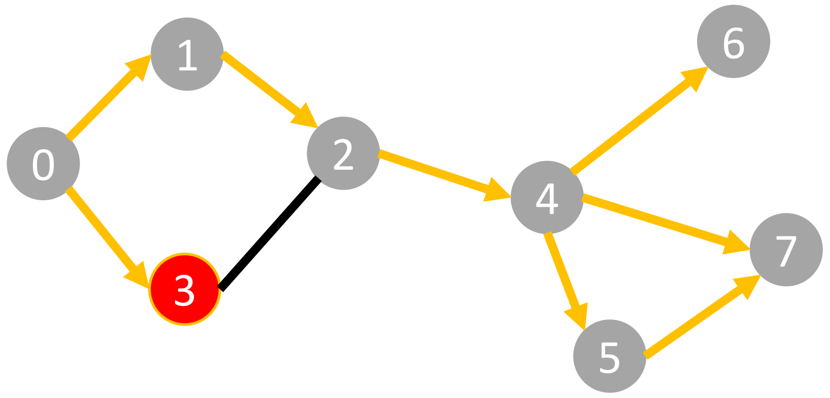


The order of visited nodes: {0,1,2,4,5,7,6}

Section 2: DFS

- Node 3 is going out of the stack

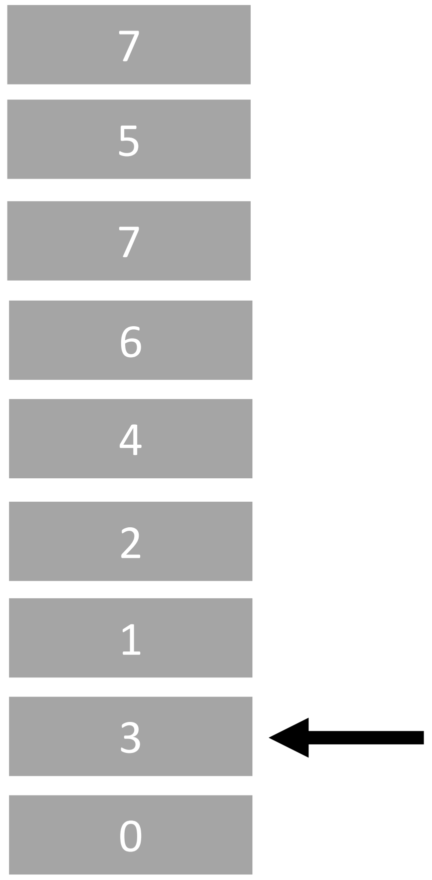


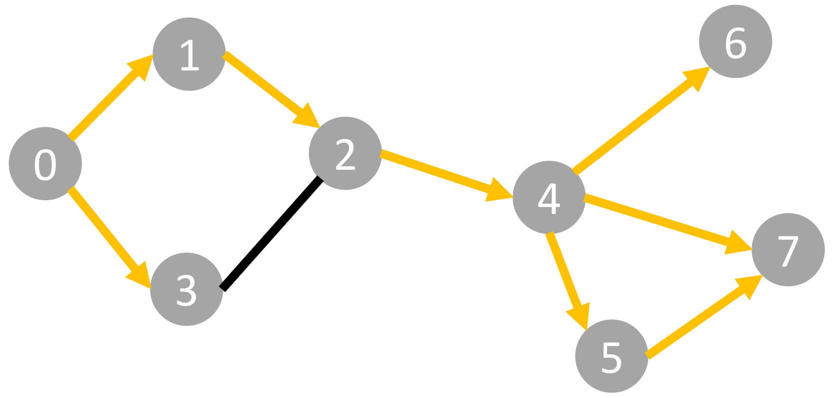


The order of visited nodes: {0,1,2,4,5,7,6}

Section 2: DFS

- Node 3 is out of the stack
- The DFS will terminate when the stack is empty

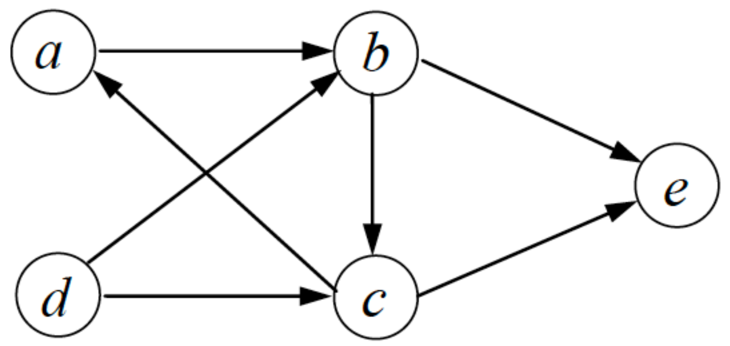




The order of visited nodes: {0,1,2,4,5,7,6,3}

Section 2: Strongly Connected Component

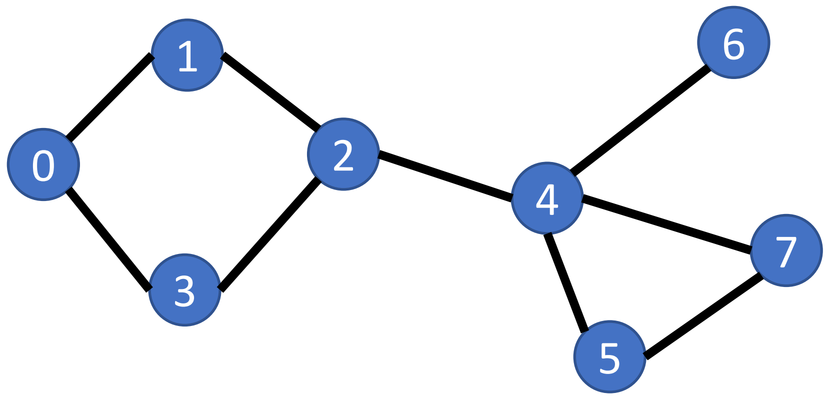
- What is a strongly connected component?
  - A strongly connected component is the portion of a directed graph in which there is a path from each node to another node.
  - It is applicable only on a directed graph.
  - Is {a,b,c} a strongly connected component?
  - Is {b,c,d} a strongly connected component?



  - How to find strongly connected components? You can refer to: https://stackoverflow.com/questions/33590974/how-to-find-strongly-connected-components-in-a-graph

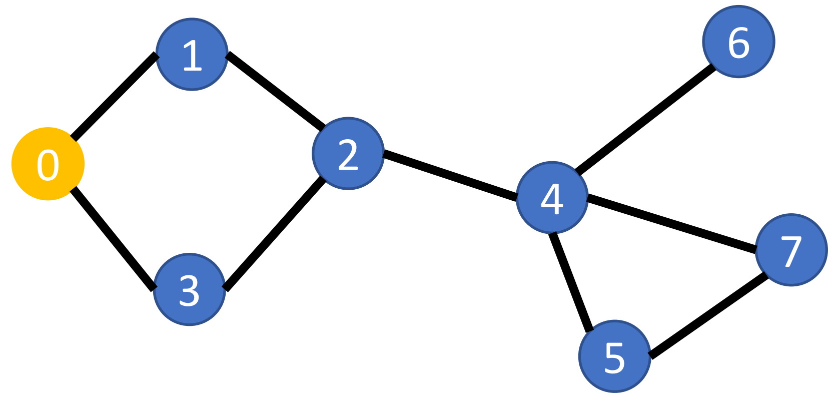
Section 2: Breadth First Search (BFS)

- The Breadth First Search (BFS) is another fundamental search algorithm used to explore nodes and edges of a graph.



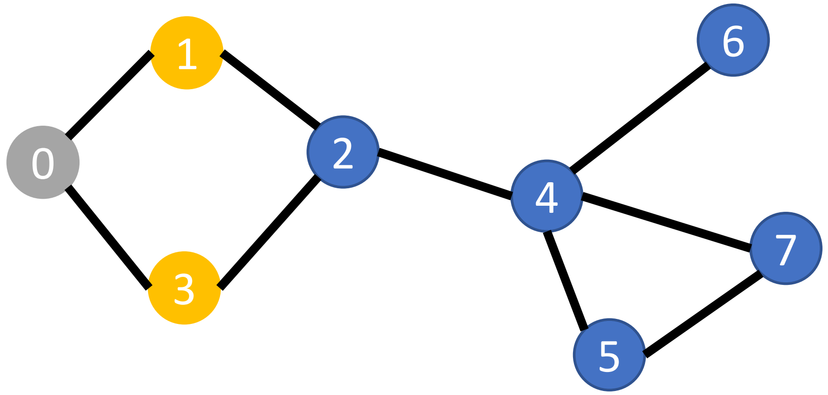
Section 2: Breadth First Search (BFS)

- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors



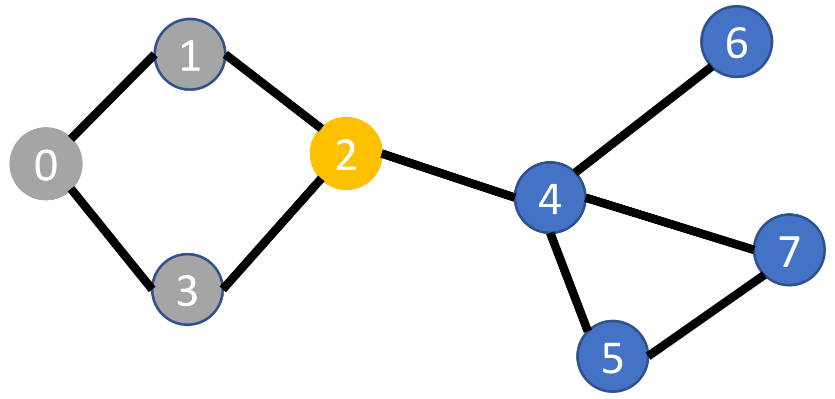
Section 2: Breadth First Search (BFS)

- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors



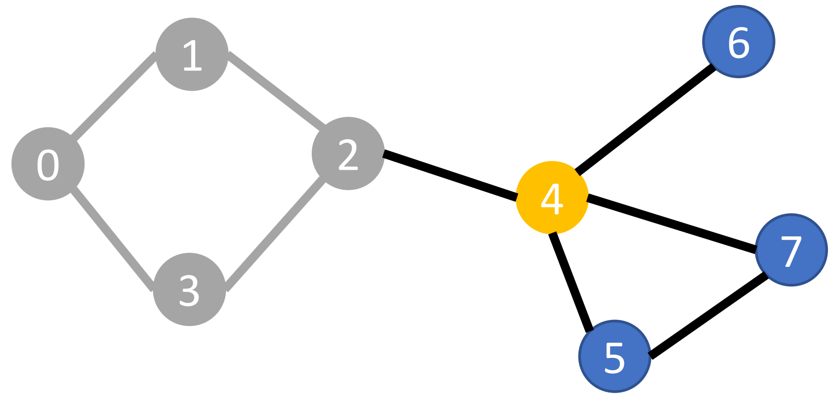
Section 2: Breadth First Search (BFS)

- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors



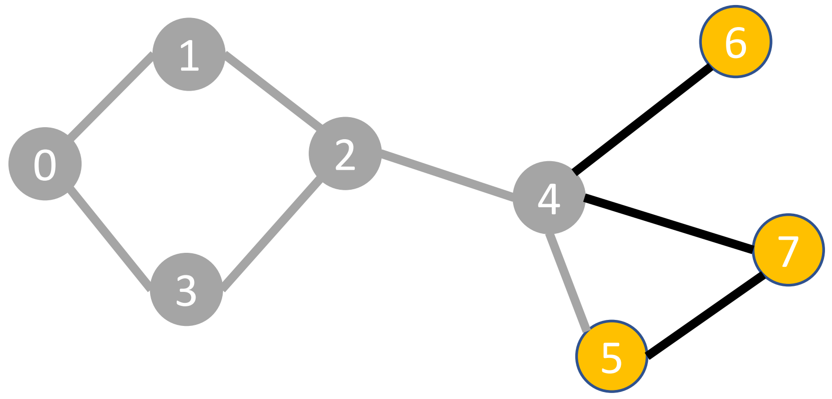
Section 2: Breadth First Search (BFS)

- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors



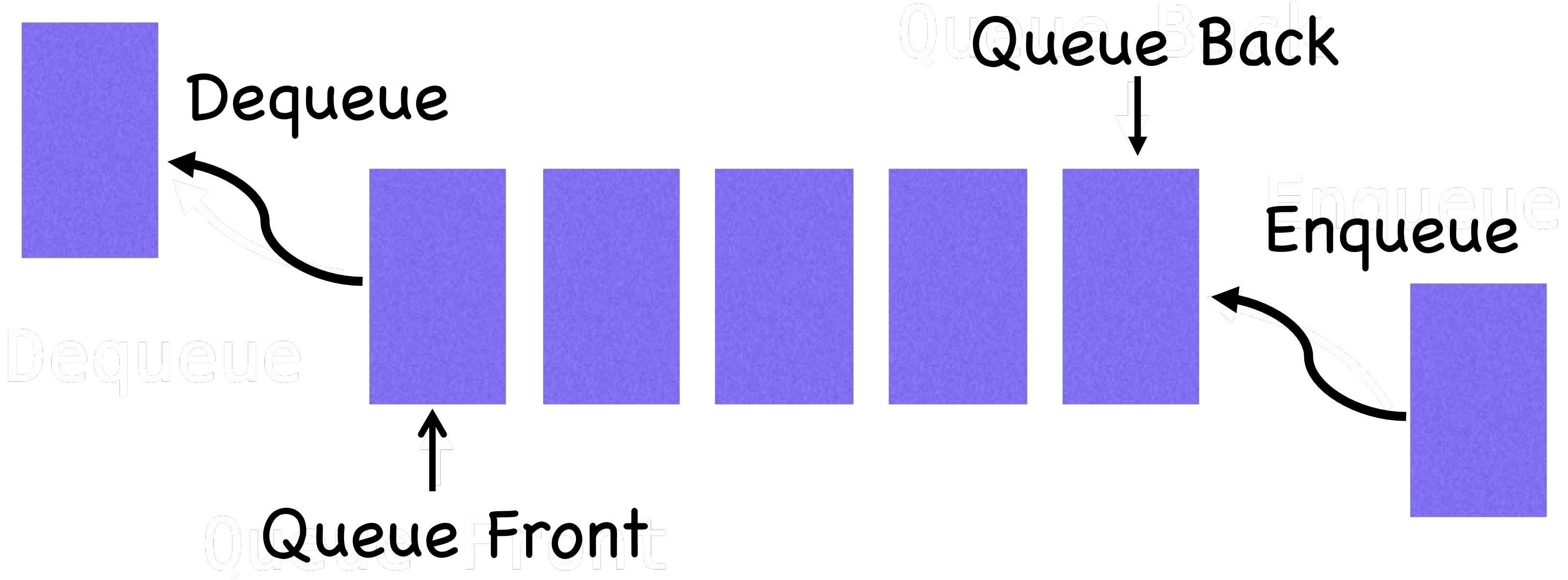
Section 2: Breadth First Search (BFS)

- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors



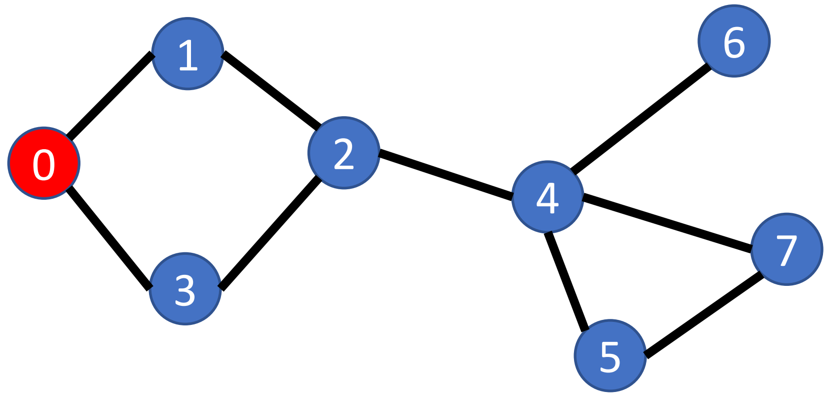
Section 2: Using a Queue

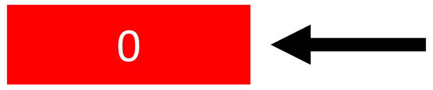
- The BFS algorithm uses a queue data structure to track which node to visit next. Upon reaching a new node the algorithm adds it the queue to visit it later. The queue data structure works just like a real-world queue such as a waiting line at a restaurant. People can either join the waiting line (enqueue) or get seated (dequeue)



Section 2: Breadth First Search (BFS)

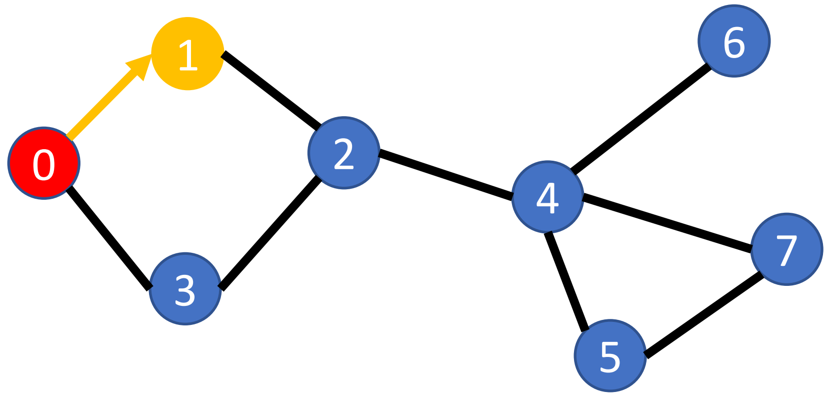
- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors

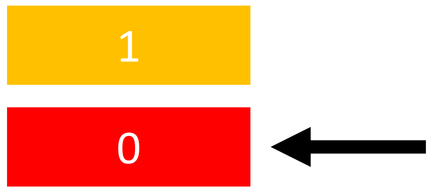




Section 2: Breadth First Search (BFS)

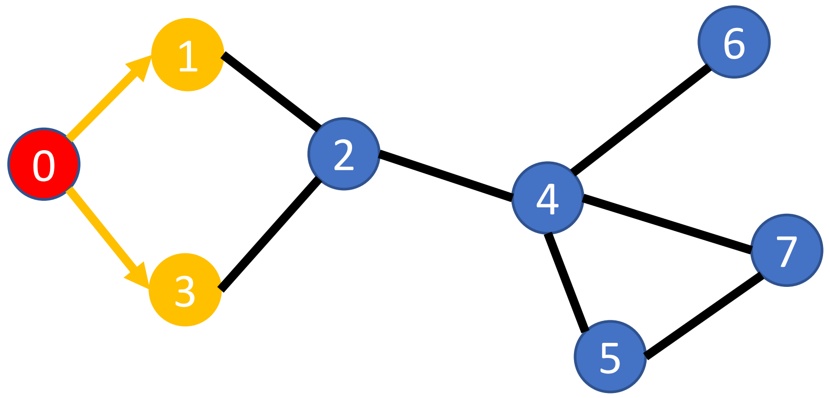
- Add the neighbors of node 1 to the queue.

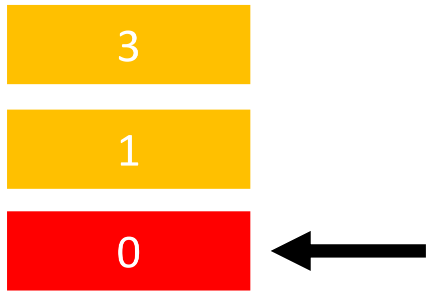




Section 2: Breadth First Search (BFS)

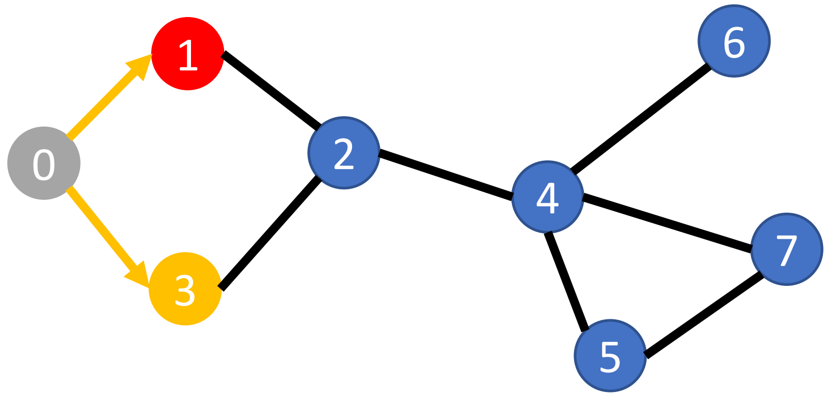
- Add the neighbors of node 1 to the queue.

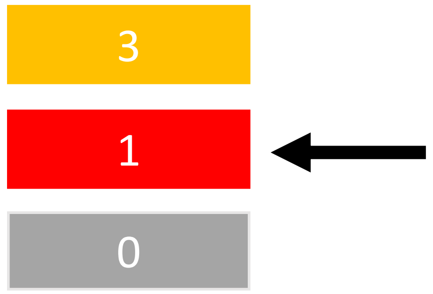




Section 2: Breadth First Search (BFS)

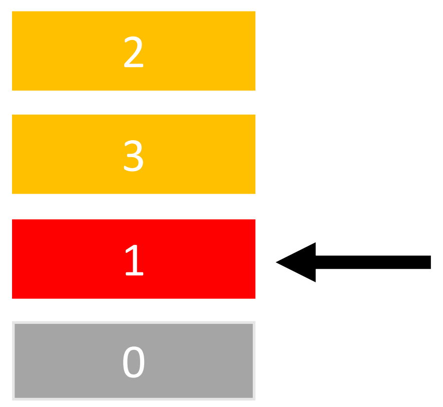
- After we add all the neighbor nodes of node 0 to the queue, we say that we have visited the node 0.
- The node 0 will dequeue.
- We will visit the next node in the queue, i..e, the node 1

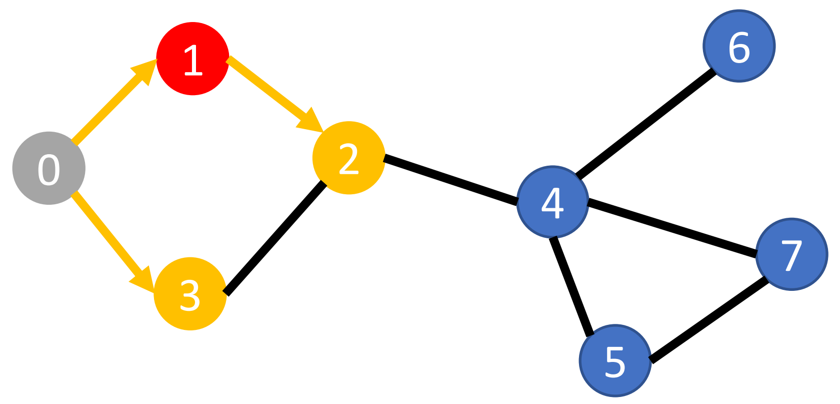




Section 2: Breadth First Search (BFS)

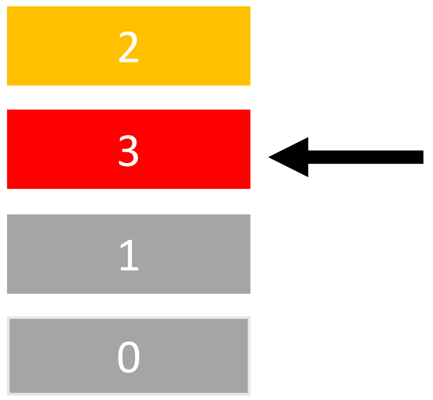
- We add the neighbor node (node 2) of node 1 to the queue.
- Node 0 has been visited, so we don't add it to the queue.

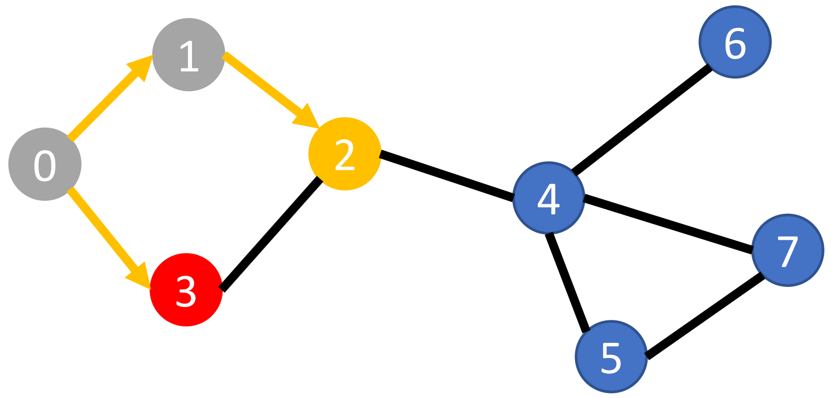




Section 2: Breadth First Search (BFS)

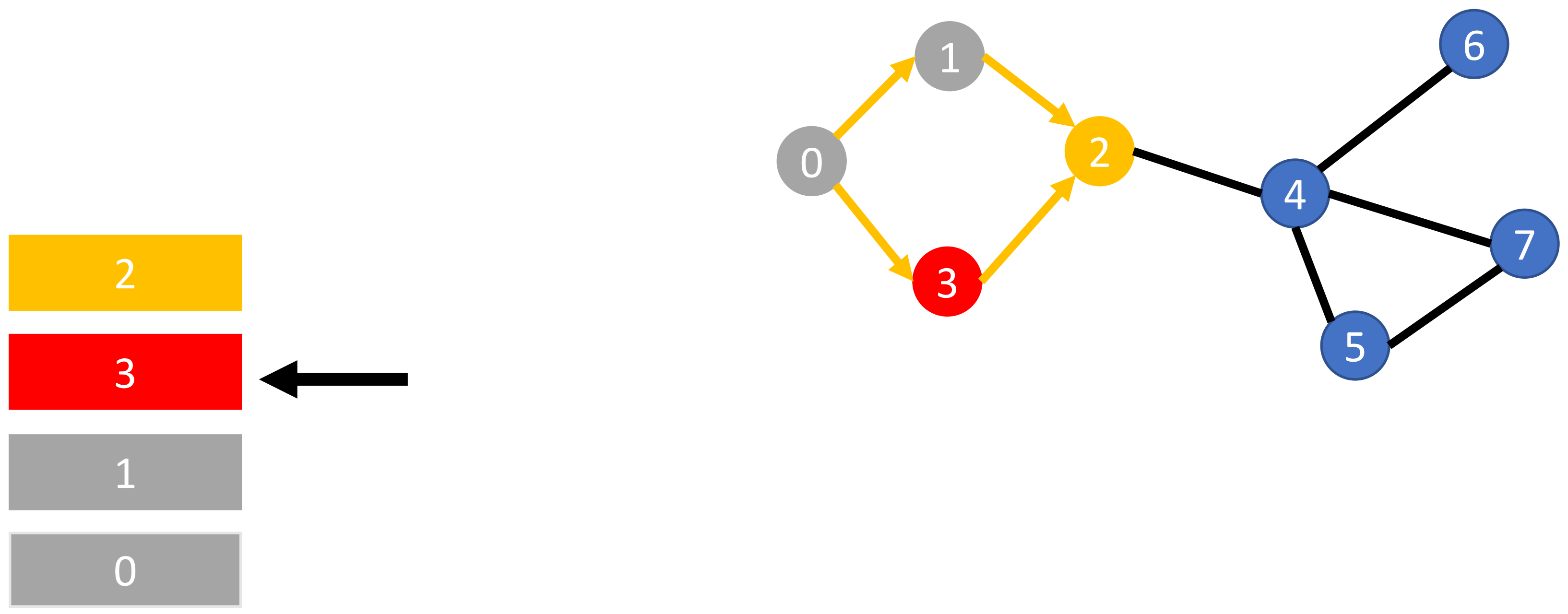
- After we add all the neighbor nodes of node 1 to the queue, we say that we have visited the node 1.
- The node 1 will dequeue.
- We will visit the next node in the queue, i..e, the node 3
- Here is the difference between DFS and BFS





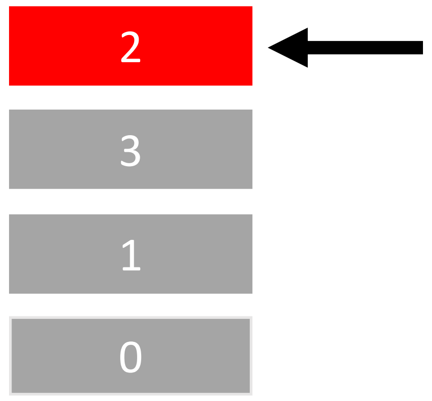
Section 2: Breadth First Search (BFS)

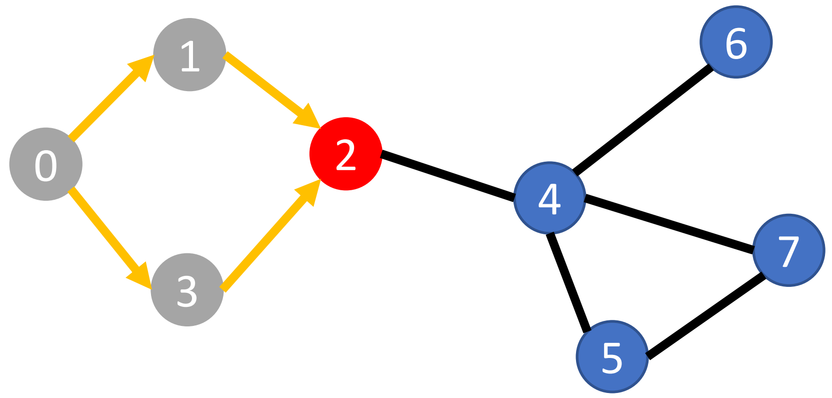
- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors



Section 2: Breadth First Search (BFS)

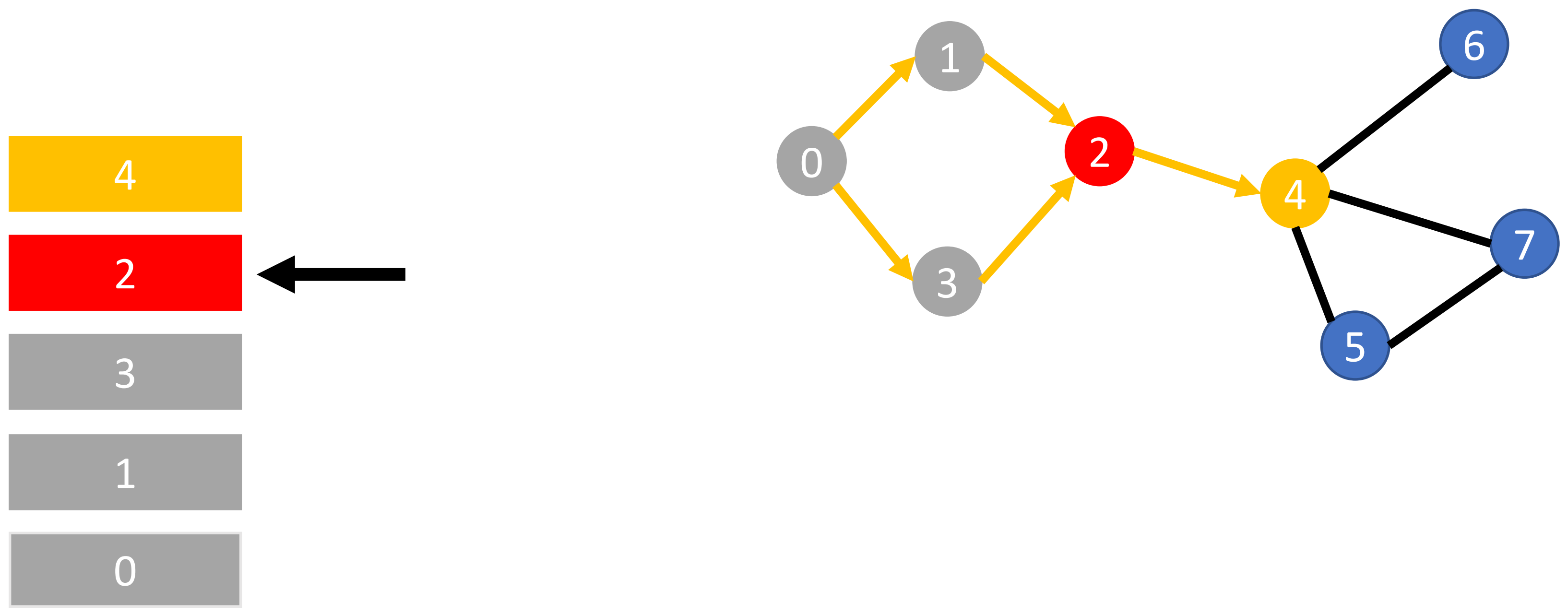
- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors





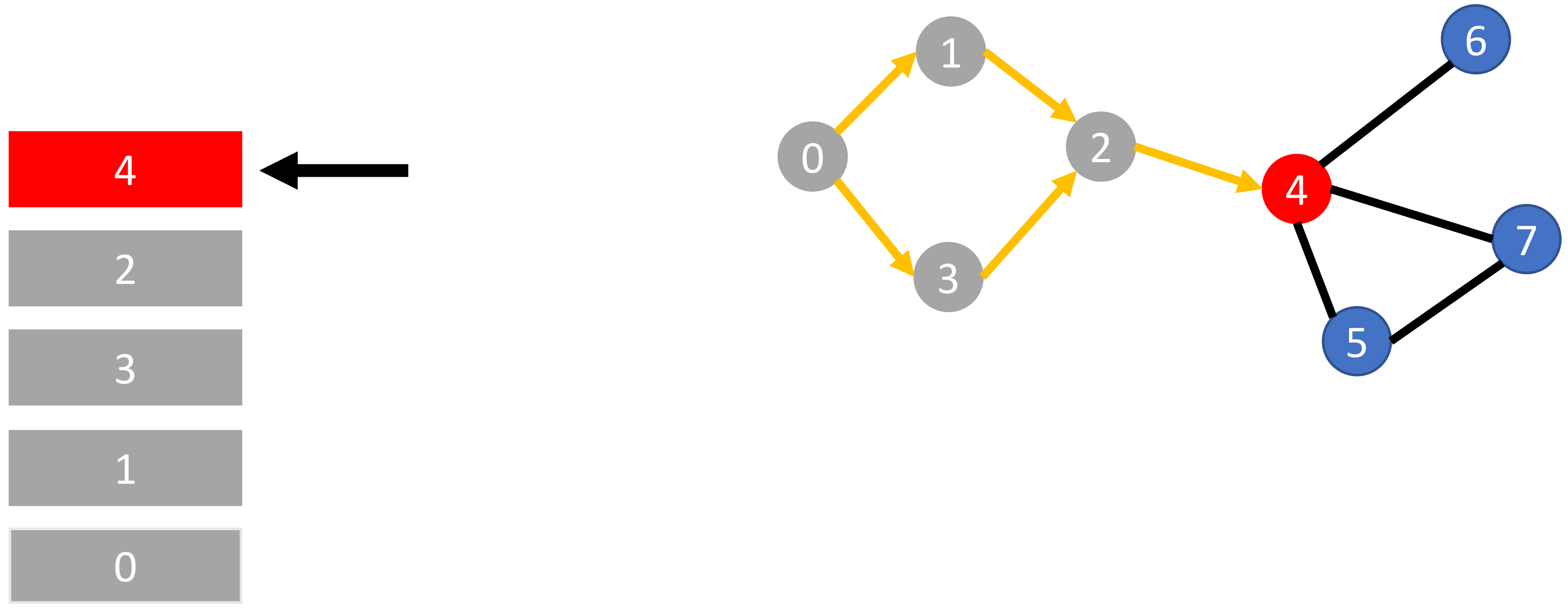
Section 2: Breadth First Search (BFS)

- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors



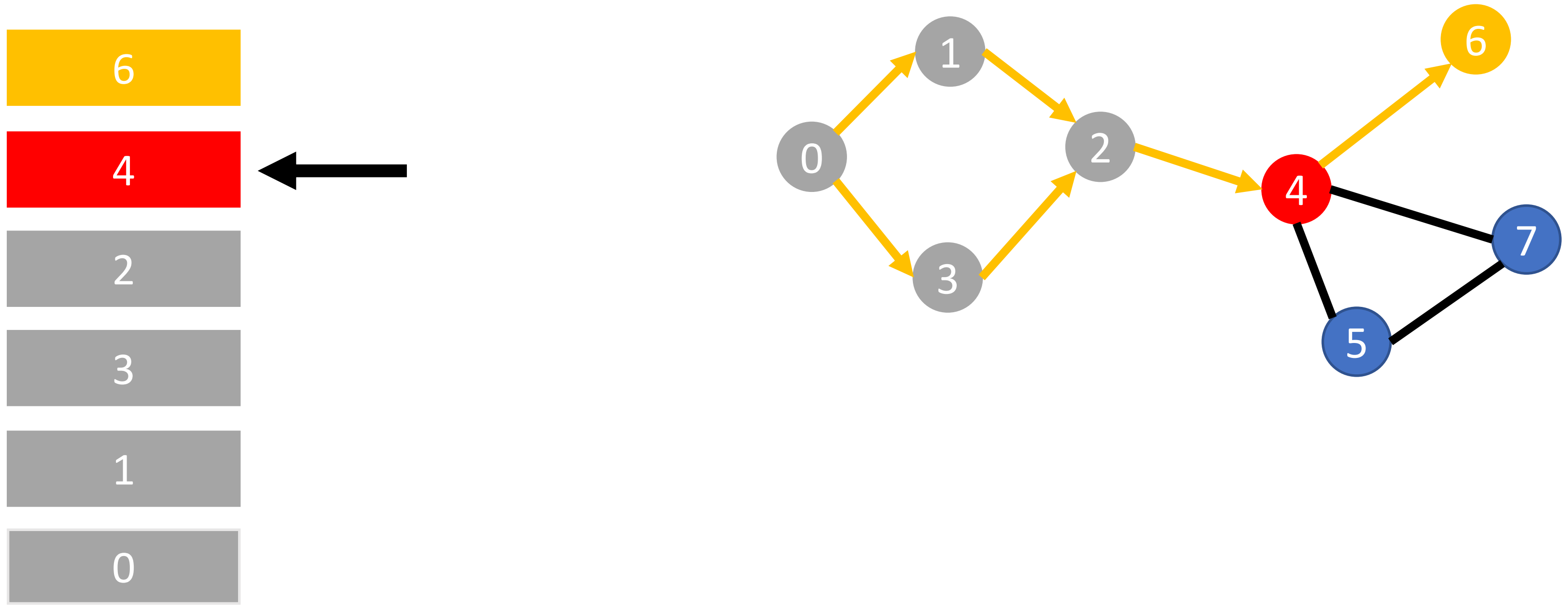
Section 2: Breadth First Search (BFS)

- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors



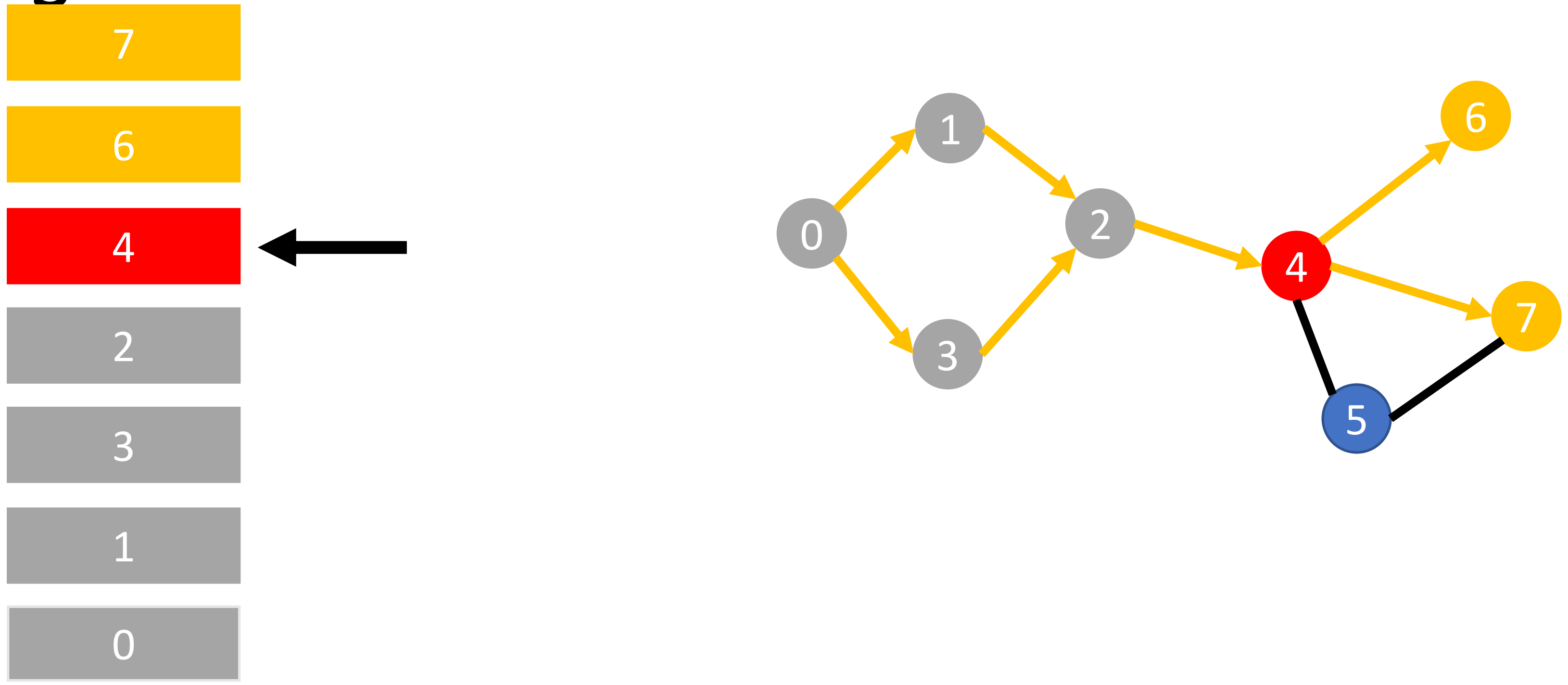
Section 2: Breadth First Search (BFS)

- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors



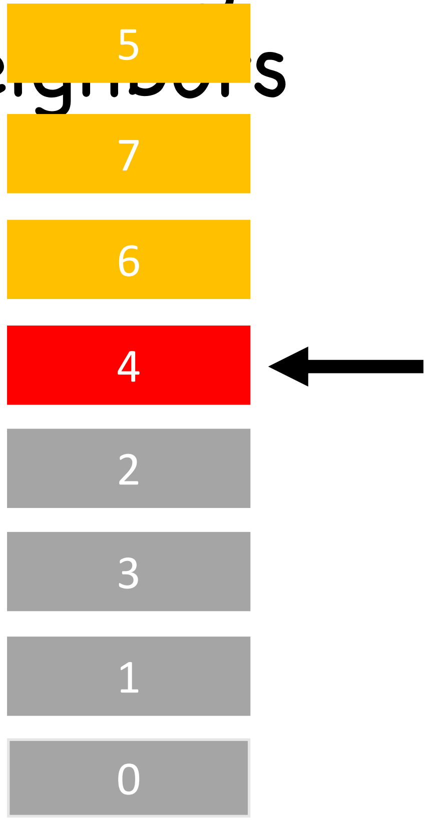
Section 2: Breadth First Search (BFS)

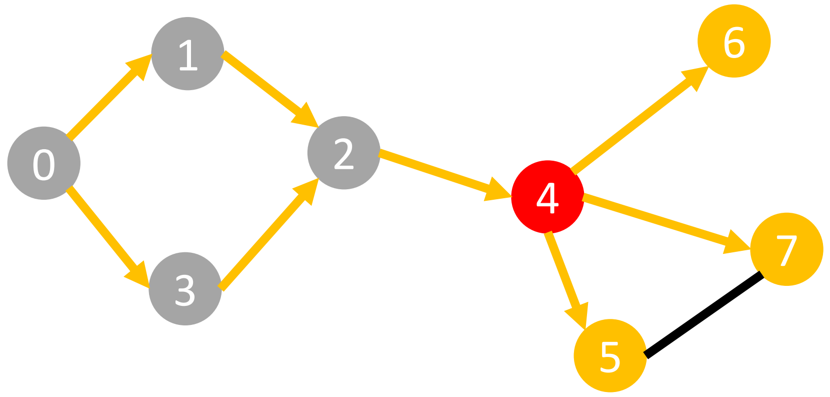
- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors



Section 2: Breadth First Search (BFS)

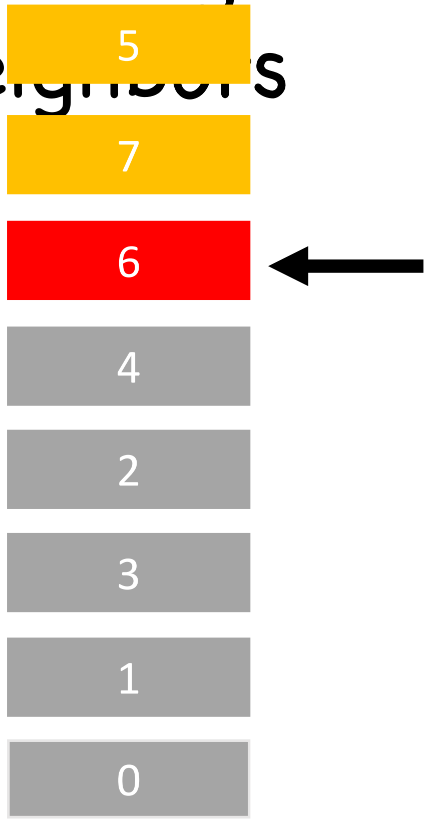
- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors

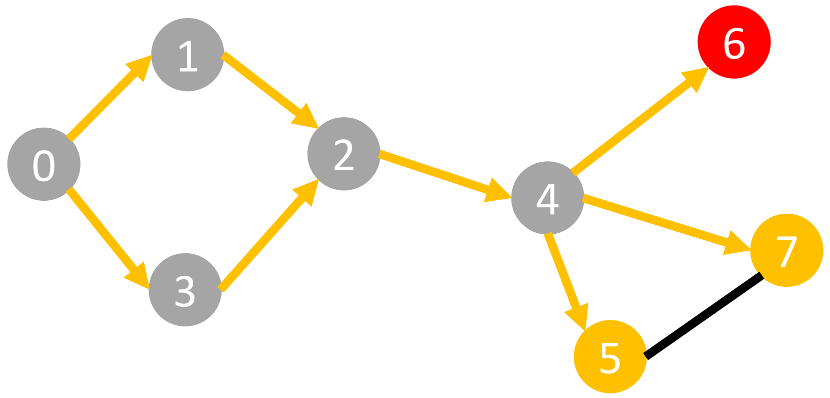




Section 2: Breadth First Search (BFS)

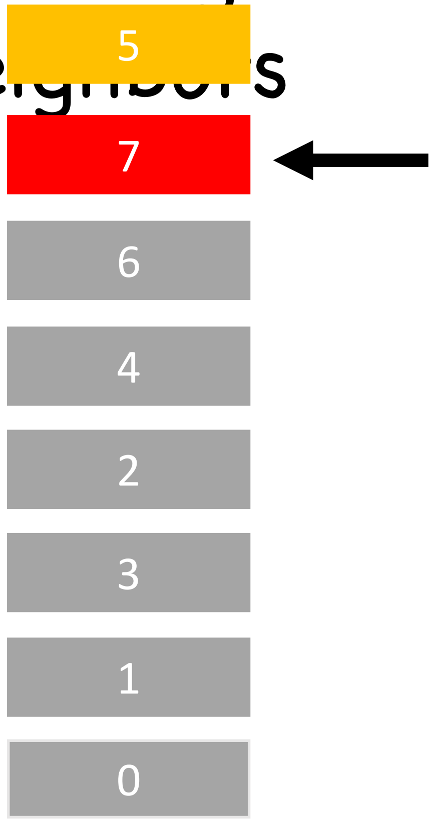
- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors

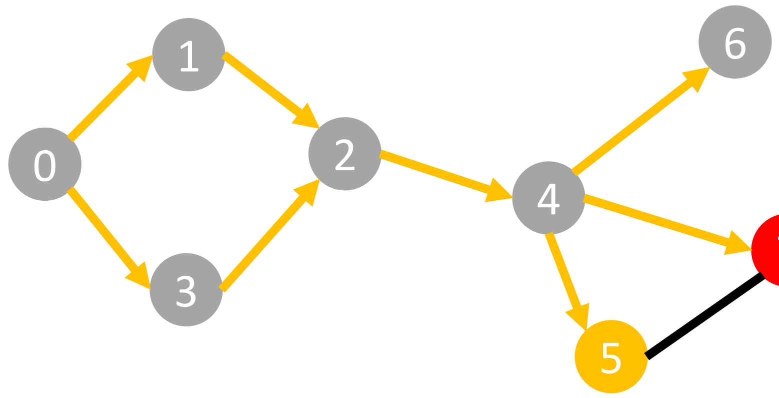




Section 2: Breadth First Search (BFS)

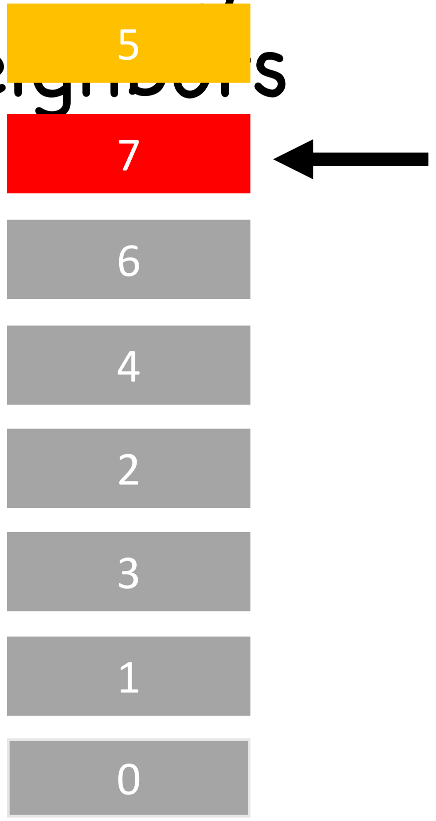
- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors

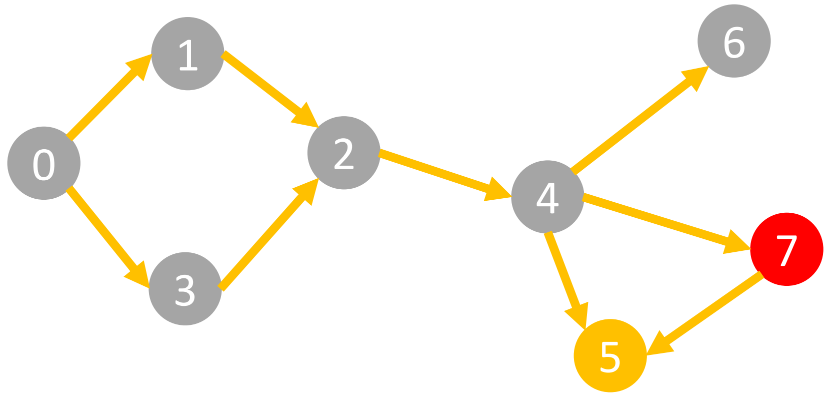




Section 2: Breadth First Search (BFS)

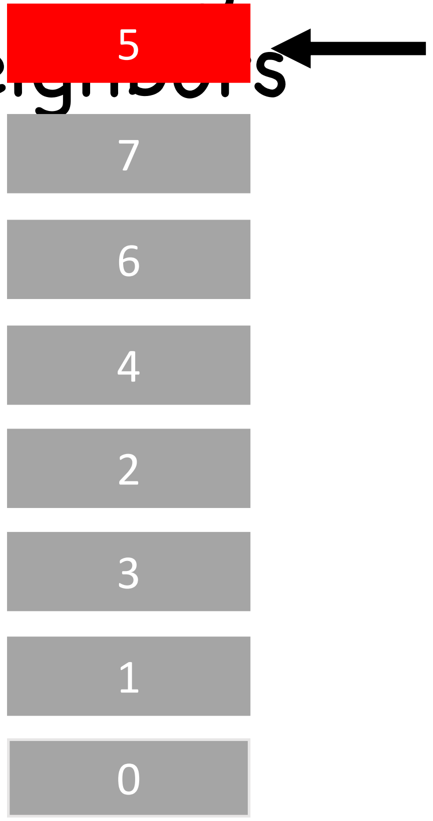
- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors

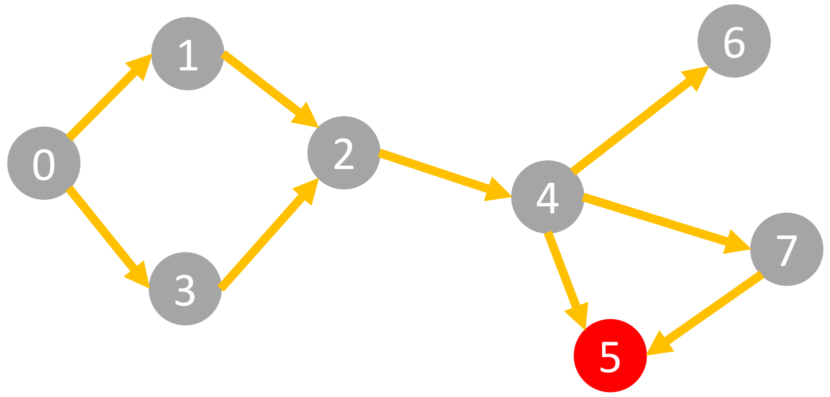




Section 2: Breadth First Search (BFS)

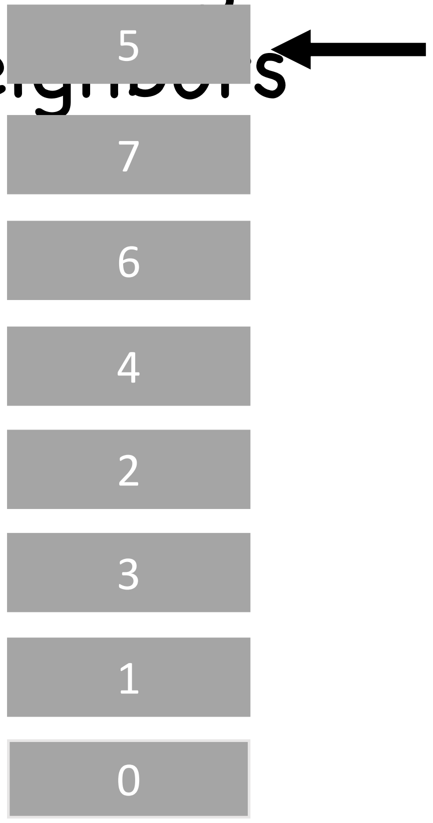
- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors

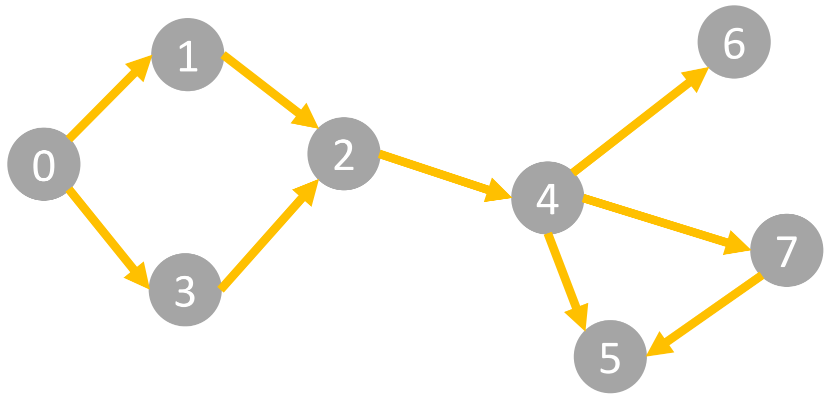




Section 2: Breadth First Search (BFS)

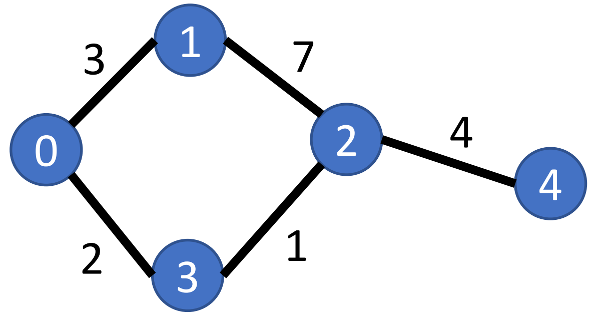
- A BFS starts at some arbitrary node of a graph and explore the neighbor nodes first, before moving to the next level neighbors





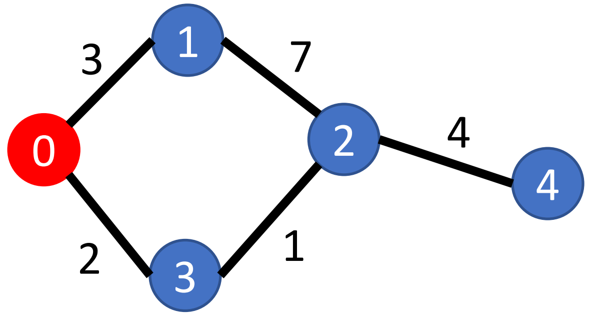
Section 2: Breadth First Search (BFS)-Application

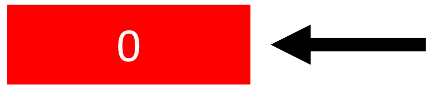
- The BFS algorithm is particularly useful for one thing: finding the shortest path on weight/unweighted graphs.
- What is the shortest path length between node 0 and node 4?



Section 2: Breadth First Search (BFS)-Application

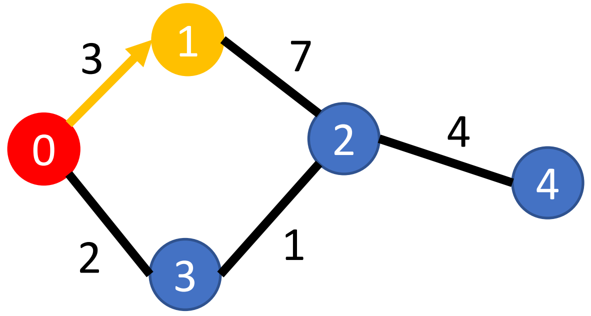
- What is the shortest path length between node 0 and node 4?

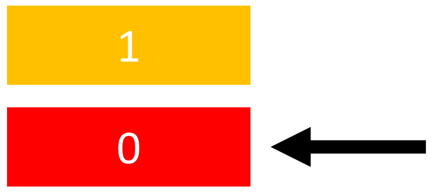




Section 2: Breadth First Search (BFS)-Application

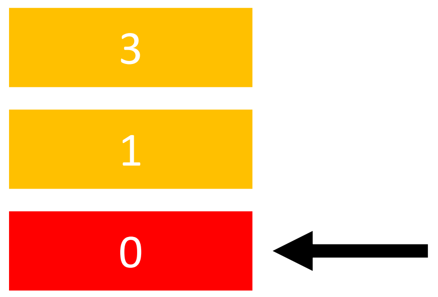
- What is the shortest path length between node 0 and node 4?

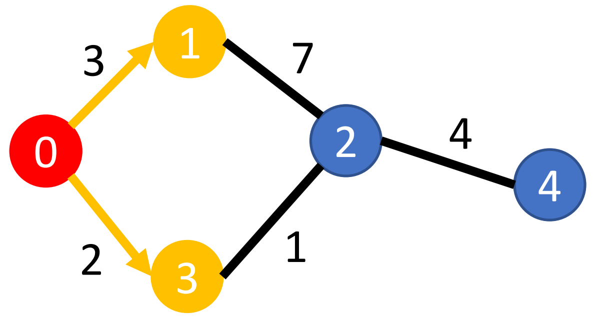




Section 2: Breadth First Search (BFS)-Application

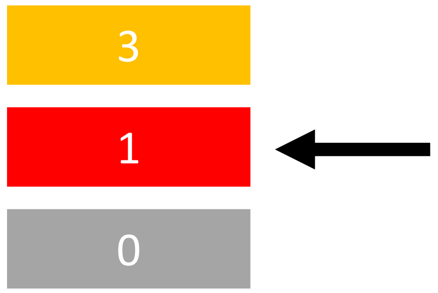
- What is the shortest path length between node 0 and node 4?



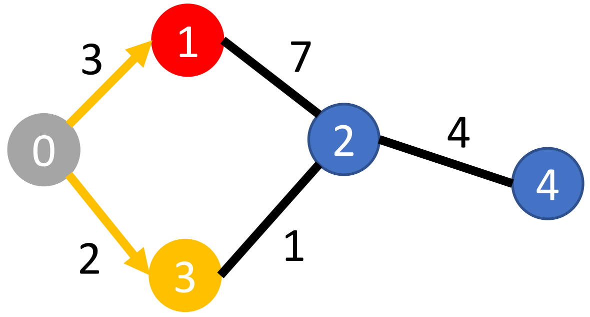


Section 2: Breadth First Search (BFS)-Application

- What is the shortest path length between node 0 and node 4?

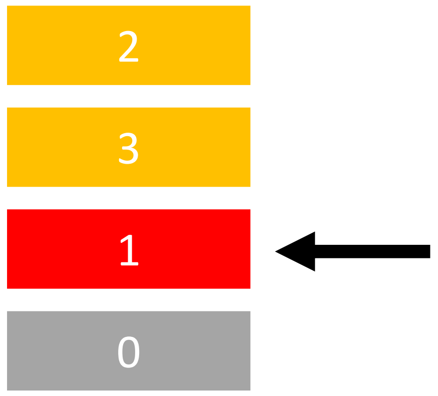


Path: 0 -->1, Length: 3

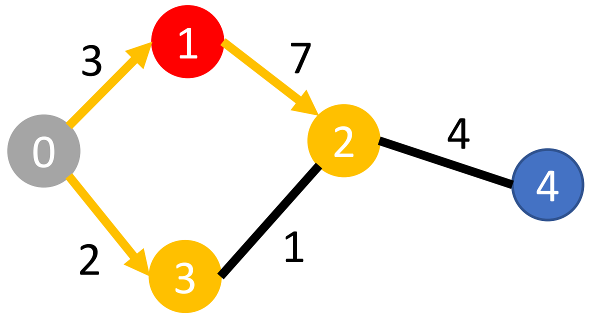


Section 2: Breadth First Search (BFS)-Application

- What is the shortest path length between node 0 and node 4?

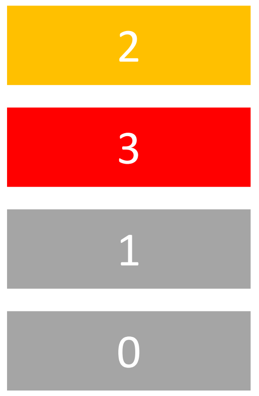


Path: 0 -->1, Length: 3



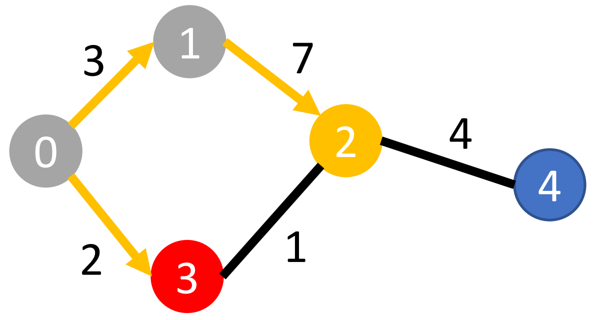
Section 2: Breadth First Search (BFS)-Application

- What is the shortest path length between node 0 and node 4?



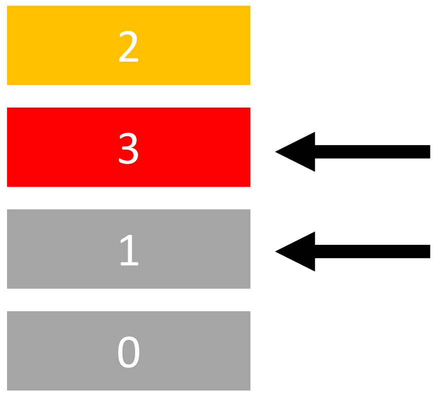
Path: 0 -->3, Length: 2

Path: 0 -->1, Length: 3



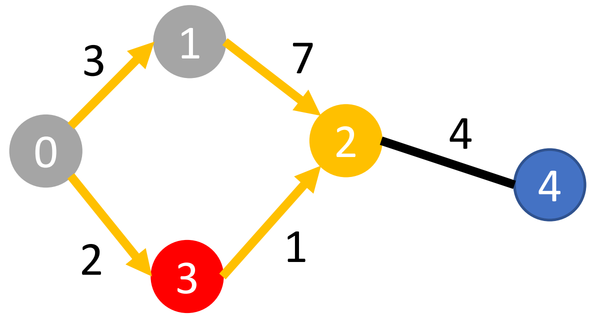
Section 2: Breadth First Search (BFS)-Application

- What is the shortest path length between node 0 and node 4?



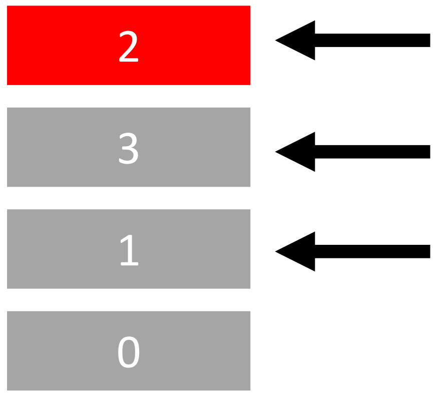
Path: 0 -->3, Length: 2

Path: 0 -->1, Length: 3



Section 2: Breadth First Search (BFS)-Application

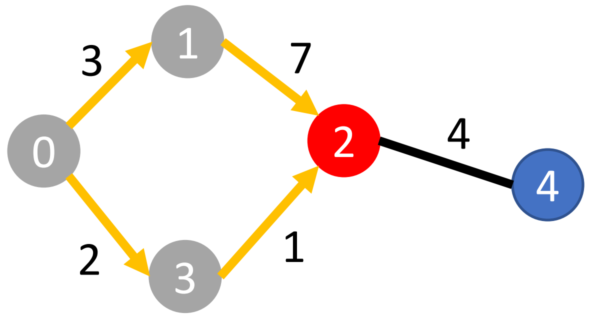
- What is the shortest path length between node 0 and node 4?



Path: 0 →1→2, Length: 3+7=10
Path: 0 →3→2, Length:2+1=3

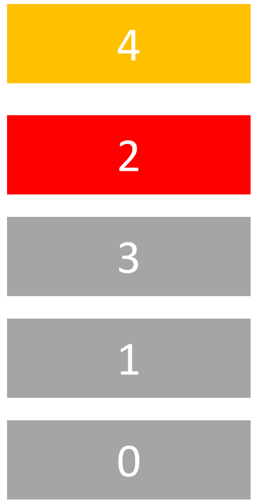
Path: 0 -->3, Length: 2

Path: 0 -->1, Length: 3



Section 2: Breadth First Search (BFS)-Application

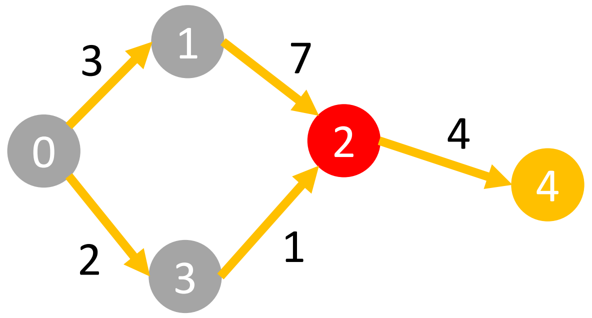
- What is the shortest path length between node 0 and node 4?



Path: 0 →1→2, Length: 3+7=10
Path: 0 →3→2, Length:2+1=3

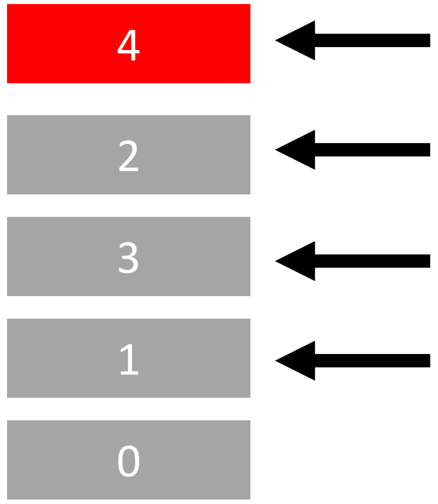
Path: 0 -->3, Length: 2

Path: 0 -->1, Length: 3



Section 2: Breadth First Search (BFS)-Application

- What is the shortest path length between node 0 and node 4?

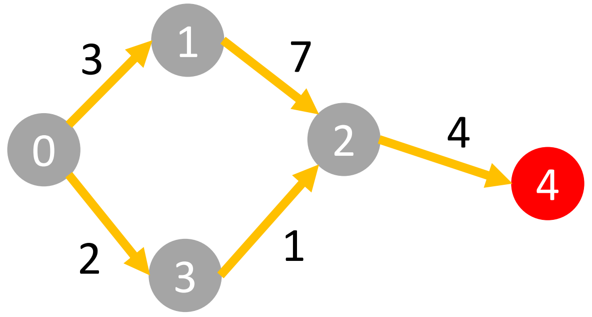


Path: 0 →1→2→4, Length: 3+7+4=14
Path: 0 →3→2→4, Length:2+1+4=7

Path: 0 →1→2, Length: 3+7=10
Path: 0 →3→2, Length:2+1=3

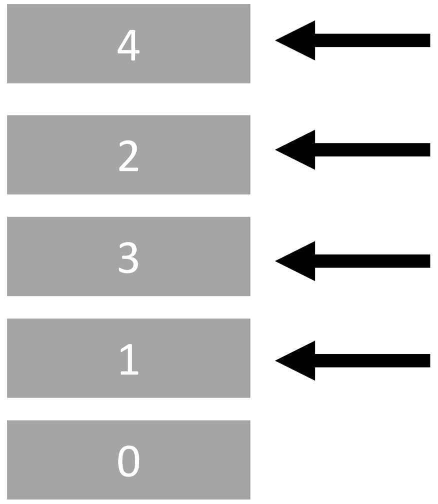
Path: 0 -->3, Length: 2

Path: 0 -->1, Length: 3



Section 2: Breadth First Search (BFS)-Application

- What is the shortest path length between node 0 and node 4?



Path: 0 →1→2→4, Length: 3+7+4=14

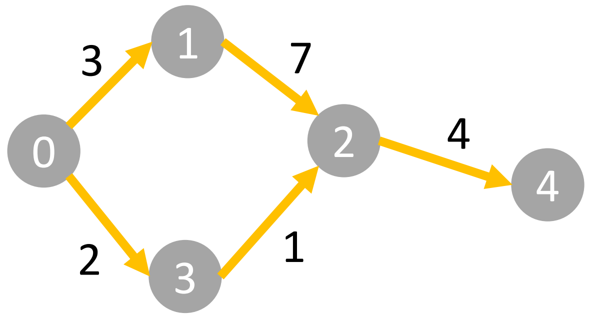
Path: 0 →3→2→4, Length:2+1+4=7

Path: 0 →1→2, Length: 3+7=10

Path: 0 →3→2, Length:2+1=3

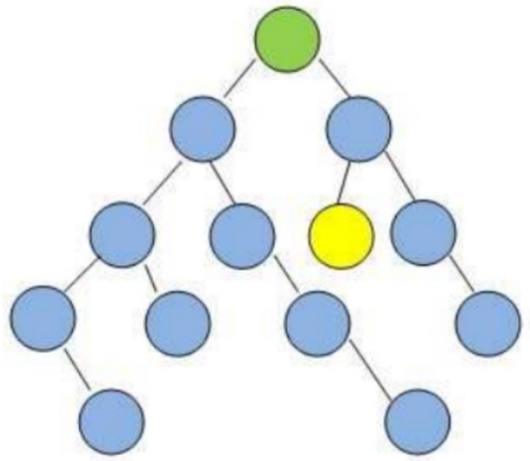
Path: 0 -->3, Length: 2

Path: 0 -->1, Length: 3



Exercise

- In the following graphs, assume that if there is ever a choice amongst multiple nodes, both the BFS and DFS algorithms will choose the left-most node first.



A: BFS

B: DFS

C: Neither BFS and DFS will ever encounter the goal node in this graph

D: BFS and DFS encounter same number of nodes before encounter the goal node

Q: Starting from the green node at the top, which algorithm will visit the least number of nodes before visiting the yellow goal node?

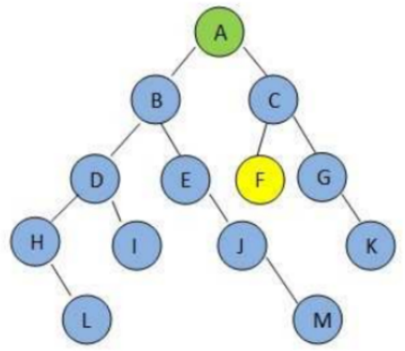
- The answer is: A (BFS)
- How can we get ?

For BFS algorithm, visiting a node's siblings before its children, while in DFS algorithm, visiting a node's children before its parents.

Before countering goal node F:

BFS algorithm encounters nodes: ABCDE

DFS algorithm encounters nodes: ABDHLIEJMC



Section 3: Coding Demo

Section3: Coding demo

- Configure your development environment:
  - Install Anaconda on Windows:
    - https://docs.anaconda.com/anaconda/install/windows/
  - Install Anaconda on MacOS:
    - https://docs.anaconda.com/anaconda/install/mac-os/
  - Install Jupyter Notebook (Optional):
    - https://docs.jupyter.org/en/latest/install/notebook-classic.html

See you next week☺

Don't forget the Online quiz 1!