CoGrammar

Welcome to this session:

Coding Interview Workshop - Graphs

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:

Ian Wyles
Designated Safeguarding Lead

Simone Botes

Nurhaan Snyman

Rafiq Manan

Ronald Munodawafa

Tevin Pitts

Scan to report a safeguarding concern

or email the Designated Safeguarding Lead:
Ian Wyles
safeguarding@hyperiondev.com

CoGrammar HyperionDev

Skills Bootcamp Coding Interview Workshop

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Skills Bootcamp Coding Interview Workshop

- For all **non-academic questions**, please submit a query: ***www.hyperiondev.com/support***

- **Report a safeguarding incident: *www.hyperiondev.com/safeguardreporting***

- We would love your feedback on lectures: **Feedback on Lectures**

- If you are hearing impaired, please kindly use your computer's function through Google chrome to enable captions.

Learning Outcomes

- ❖ Define and illustrate the fundamental concepts of graphs, including vertices (nodes), edges, directed vs. undirected graphs, weighted vs. unweighted edges, and the representation of graphs.
- ❖ Implement graph traversal algorithms in Python, focusing on depth-first search (DFS) and breadth-first search (BFS), to explore nodes and their connections.
- ❖ Apply graphs in Python to solve problems such as finding the shortest path, detecting cycles.

Lecture Overview

→ Introduction to Graph Fundamentals

→ Types of Graphs

→ Graph Traversals

→ Implementation of Graphs

CoGrammar

What is a graph in Computer Science?

A. A visual representation of data

B. A data structure that consists of nodes and edges

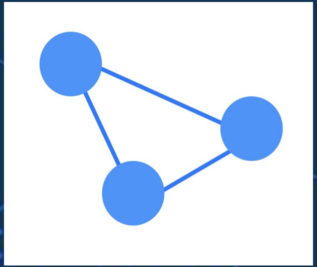
C. An algorithm that helps in the sorting of data

D. A concept which involves organising data in rows and columns

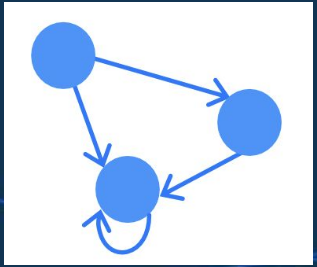
CoGrammar

Which of the following graphs is a directed graph?

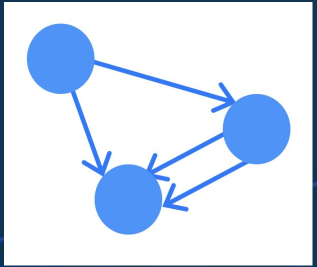
A. 1 – 2,
2 – 3,
3 - 1



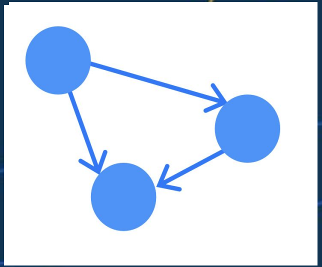
B. 1 → 2,
2 → 3,
3 → 3
3 ← 1



C. 1 → 2,
2 → 3,
3 ← 2,
1 → 3



D. 1 → 2,
2 → 3,
1 → 3



Security in Networks

Consider a network of devices and systems which you have been tasked with securing by detecting potential vulnerabilities. New devices can be added to the network at any point and connections between devices need to be made without compromising security.

- ➢ What **data structure** could be used to represent a network?
- ➢ How could we use this structure to be able to detect **potential vulnerabilities in its structure** and **finding the quickest route to transfer encrypted data**?

Modelling networks

- **Can we graphically represent a network?**
  - Use shapes to represent the devices
  - Use lines connecting the shapes to represent the connections

- **Let's try and draw this together**
  - Can we determine the number of devices?
  - Can we determine the number of connections of a device?
  - What is the shortest connection between two devices?

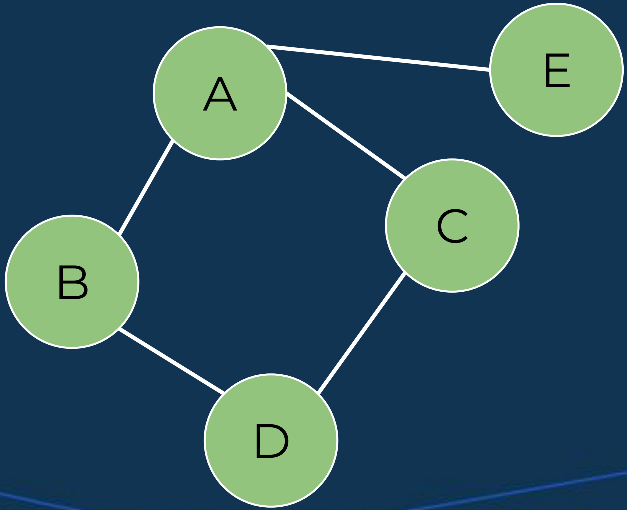
Graphs

A non-linear data structure made up of vertices or nodes and connected by edges or arcs, that is used to represent complex relationships between objects.

- Graphs are made up of two sets, the **Vertices (*V*)** and **Edges (*E*)**.
- Each element of *E* is a **pair** consisting of two elements from *V*.
- Vertices can be **labelled** and may be a **reference** to an external entity with additional information known as **attributes**.
- Edges can be **labelled** and the pairs can be **ordered** depending on the type of the graph.

CoGrammar

- Graphs are depicted visually using circles or boxes to represent vertices, and lines (or arrows) between the circles or boxes to represent edges. This can only be done for small datasets.

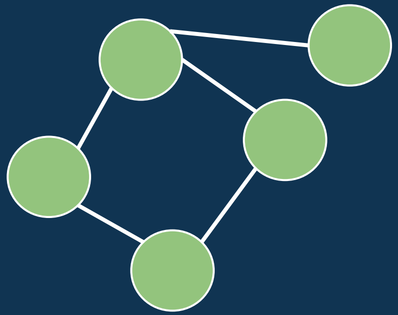
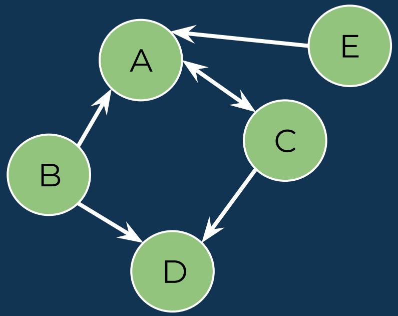


- **Neighbours:** Two nodes that are connected by an edge. This property is also known as **adjacency**.
  - *E.g. A is adjacent to B, A and B are neighbours*
- **Degree:** The number of other nodes that a node is connected to (i.e. the number of neighbours a node has).
  - *E.g. The degree of A is 3*
- **Loop:** An edge that connects a node to itself.

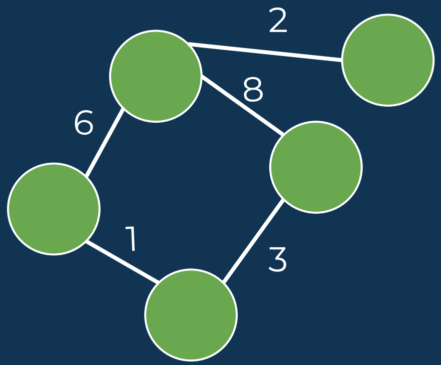
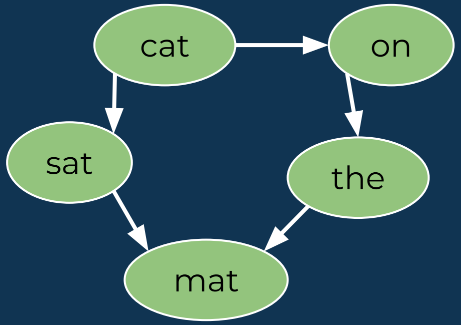
CoGrammar

- ❖ **Path:** A sequence of nodes that are connected by edges.
  - ➢ *E.g. (A, B, D) denoted using **sequence notation ()***
- ❖ **Cycle:** A closed path which starts and ends at the same node and no node is visited more than once.
  - ➢ *E.g. {A, C, D, B, A}*

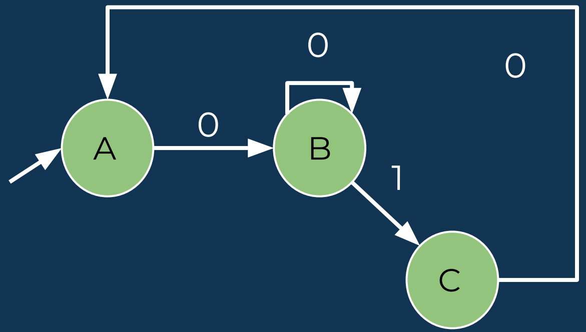
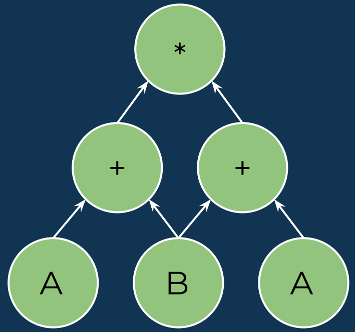
Types of Graphs

Undirected Graphs Edges connecting nodes have no direction. For this graph, the order of the pairs of vertices in the edge set does not matter. **Applications:** Social networks, Recommendation Systems	
Directed Graphs Edges connecting nodes have specified directions. Edges may also be bidirectional. This graph cannot contain any loops. **Applications:** Maps, Network Routing, WWW	

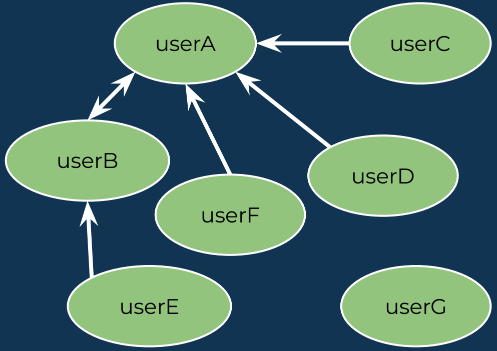
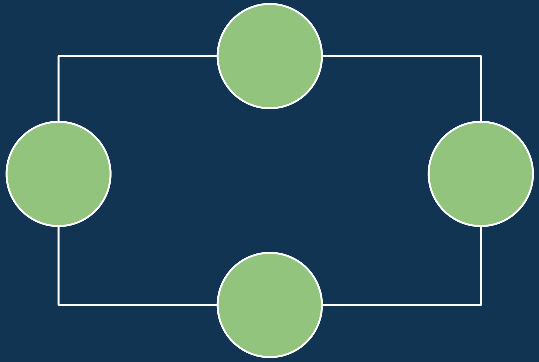
CoGrammar

Weighted/Labelled Graphs Directed/Undirected graphs that have values associated with each of its edges. These values can record any information relating to the edge e.g. distance between nodes. **Applications:** Transportation Networks, Financial/Transactional Networks	
Vertex Labelled Graphs Directed/Undirected graphs where the vertices/nodes in the graph are labelled with information which identifies the vertex. **Applications:** Biological Networks (molecular structures), Semantic Networks	

CoGrammar

Cyclic Graphs A directed graph which contains at least one cycle. **<u>Applications:</u>** Task Scheduling, Manufacturing Processes, Finite State Machines (a mathematical model of computation)	
Directed Acyclic Graphs Also known as a DAG. A directed graph with no cycles. Various use-cases across fields. **<u>Applications:</u>** Dependency Resolution Systems e.g. package management, Project Management, Compiler Design	

CoGrammar

Disconnected Graphs The graph contains nodes which are not connected via an edge to any other nodes in the graph. **Applications:** Social Networks, Transportation Networks, Component Analysis	
Connected Graphs There is a path from any node in the graph to any other node in the graph. **Applications:** Communication Networks, Routing Algorithms, Circuit Design, Data Analysis	

CoGrammar

Graph Traversal

- The process of **visiting each node in a graph exactly once**
- Types of graph traversal algorithms:
  - **Depth-First Search (DFS):** How to explore as deep as possible for each branch in a graph, before backtracking to explore an alternative branch.
  - **Breadth-First Search (BFS):** Level-by-level exploration of nodes.

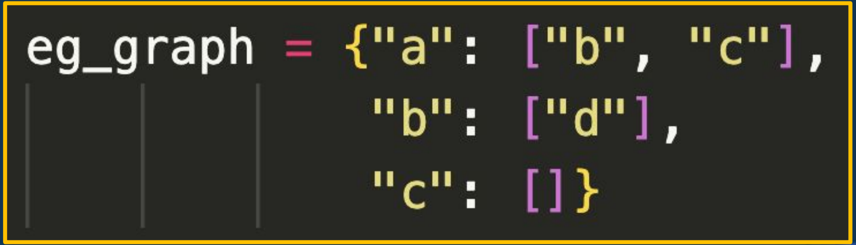
Let’s Breathe!

Let’s take a small break before moving on to the next topic.

CoGrammar

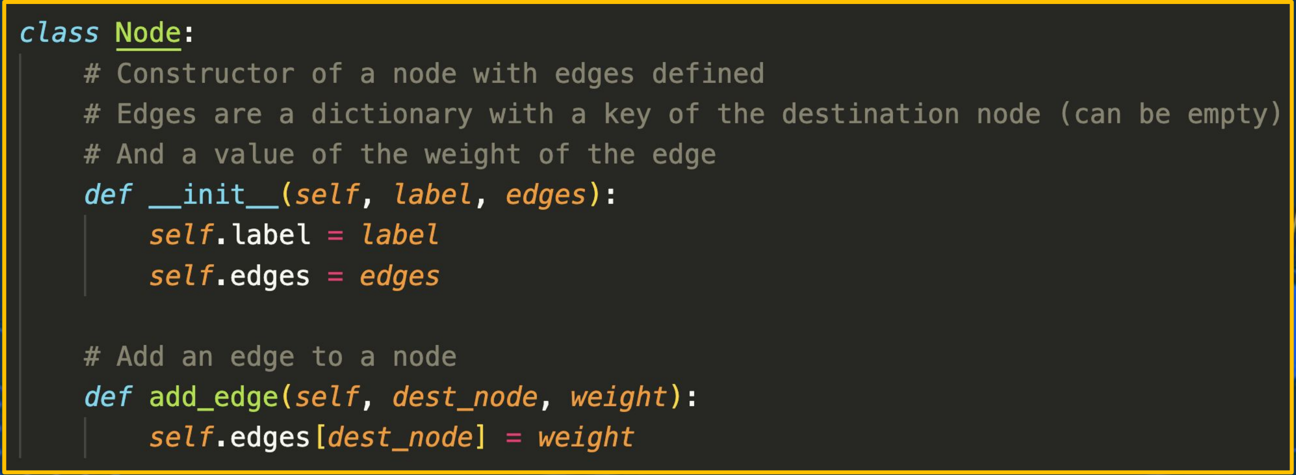
Implementation of Graphs

- ❖ The simplest way to implement a graph is using **dictionaries**.
- ❖ This method can be used for undirected graphs and edge-weighted graphs. Can anyone guess how we can do that given the implementation below?
- ❖ We can use **recursive functions** to search for paths in the graph.



```
eg_graph = {"a": ["b", "c"],
            "b": ["d"],
            "c": []}
```

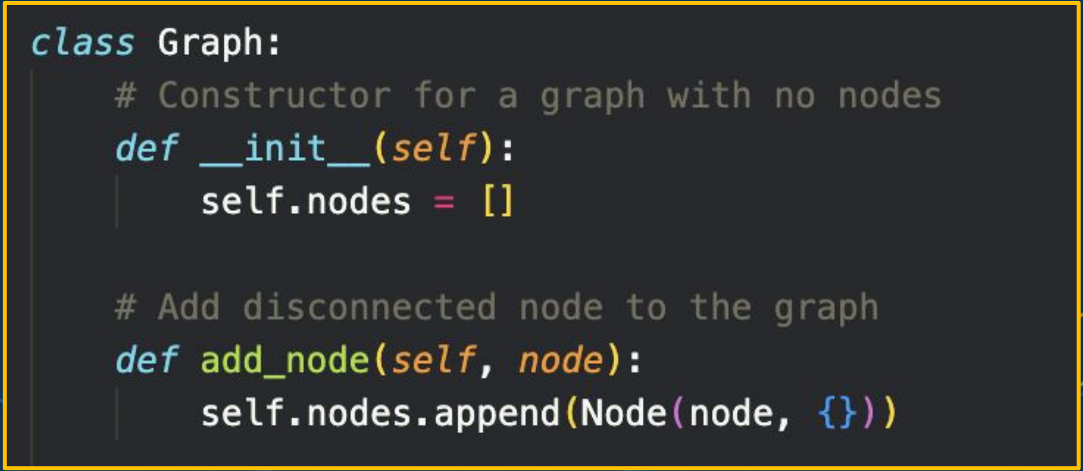
- For a more complicated implementation with a little more control over the structure and implementation, an **OOP approach** can be taken.
- We create a **Node class** which stores the data in the node and the weights of all the connected nodes.



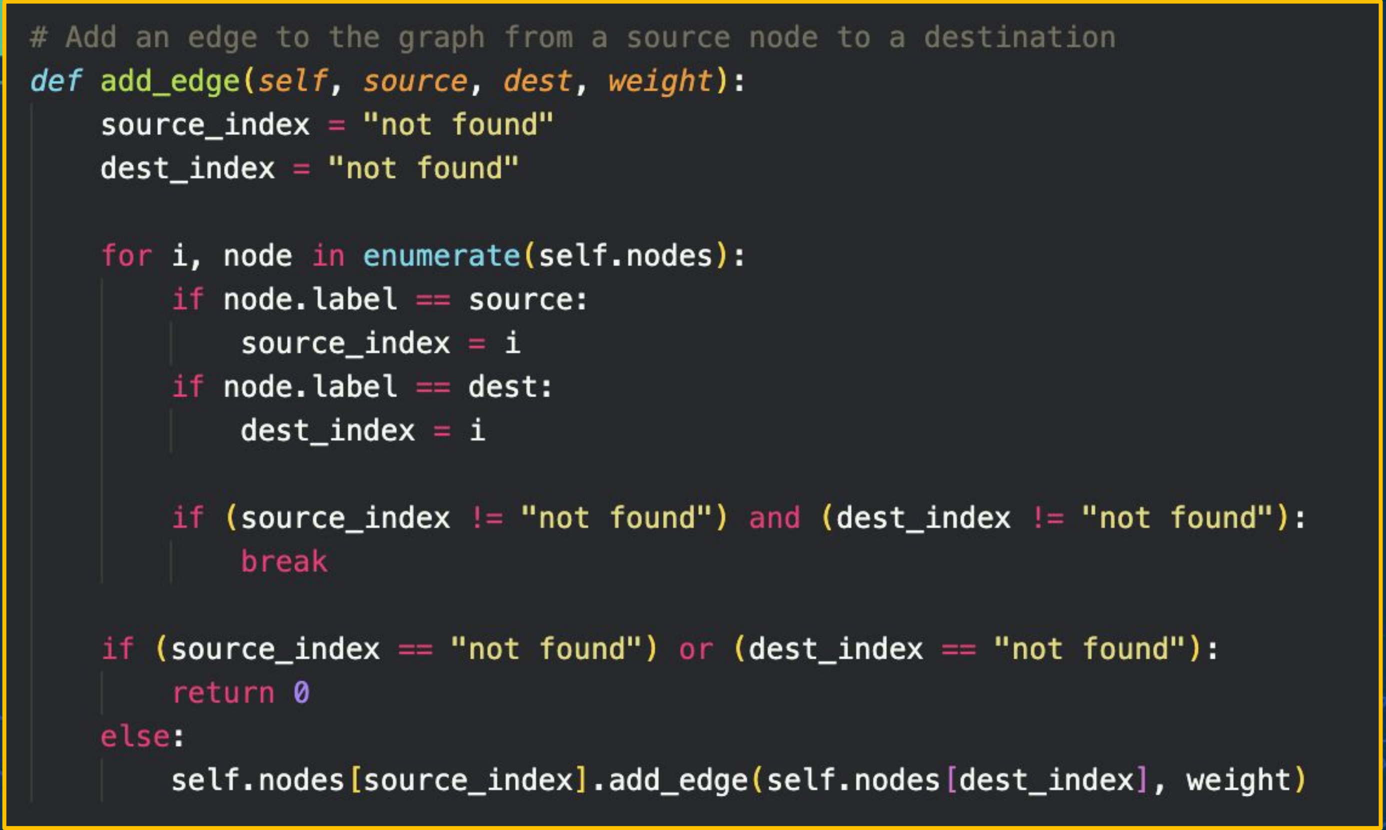
```python
class Node:
    # Constructor of a node with edges defined
    # Edges are a dictionary with a key of the destination node (can be empty)
    # And a value of the weight of the edge
    def __init__(self, label, edges):
        self.label = label
        self.edges = edges

    # Add an edge to a node
    def add_edge(self, dest_node, weight):
        self.edges[dest_node] = weight
```

CoGrammar

- We create a **Graph class** to store all the nodes in the Graph.
- In this class we add functions to **add nodes**, **add edges**, **print out the graph** and **any additional functions (like searching, sorting etc).**



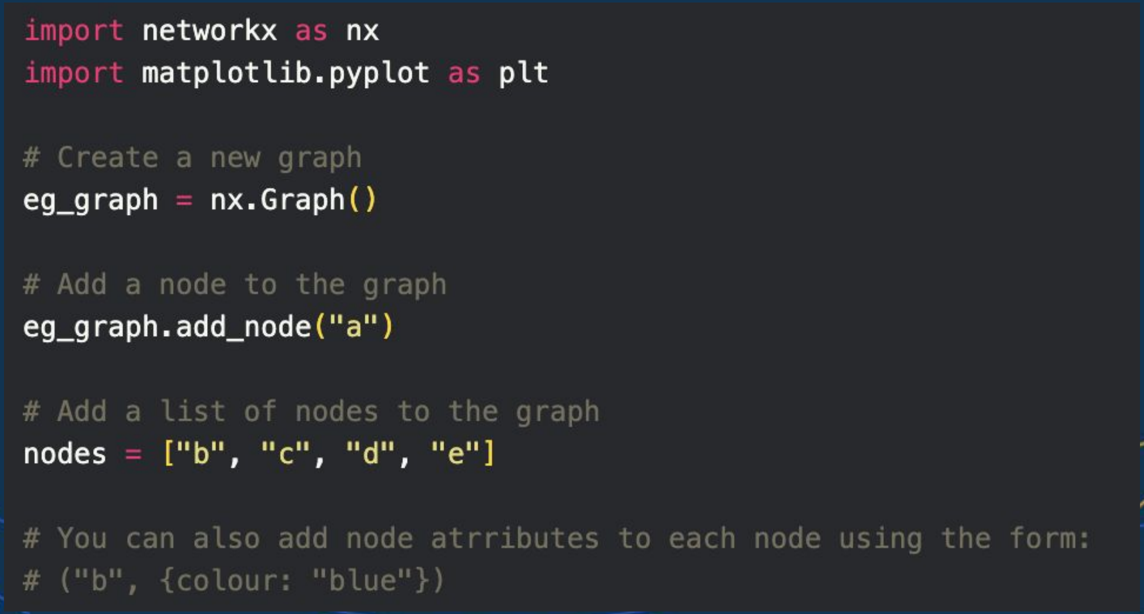
```python
class Graph:
    # Constructor for a graph with no nodes
    def __init__(self):
        self.nodes = []

    # Add disconnected node to the graph
    def add_node(self, node):
        self.nodes.append(Node(node, {}))
```



```python
# Add an edge to the graph from a source node to a destination
def add_edge(self, source, dest, weight):
    source_index = "not found"
    dest_index = "not found"

    for i, node in enumerate(self.nodes):
        if node.label == source:
            source_index = i
        if node.label == dest:
            dest_index = i

        if (source_index != "not found") and (dest_index != "not found"):
            break

    if (source_index == "not found") or (dest_index == "not found"):
        return 0
    else:
        self.nodes[source_index].add_edge(self.nodes[dest_index], weight)
```

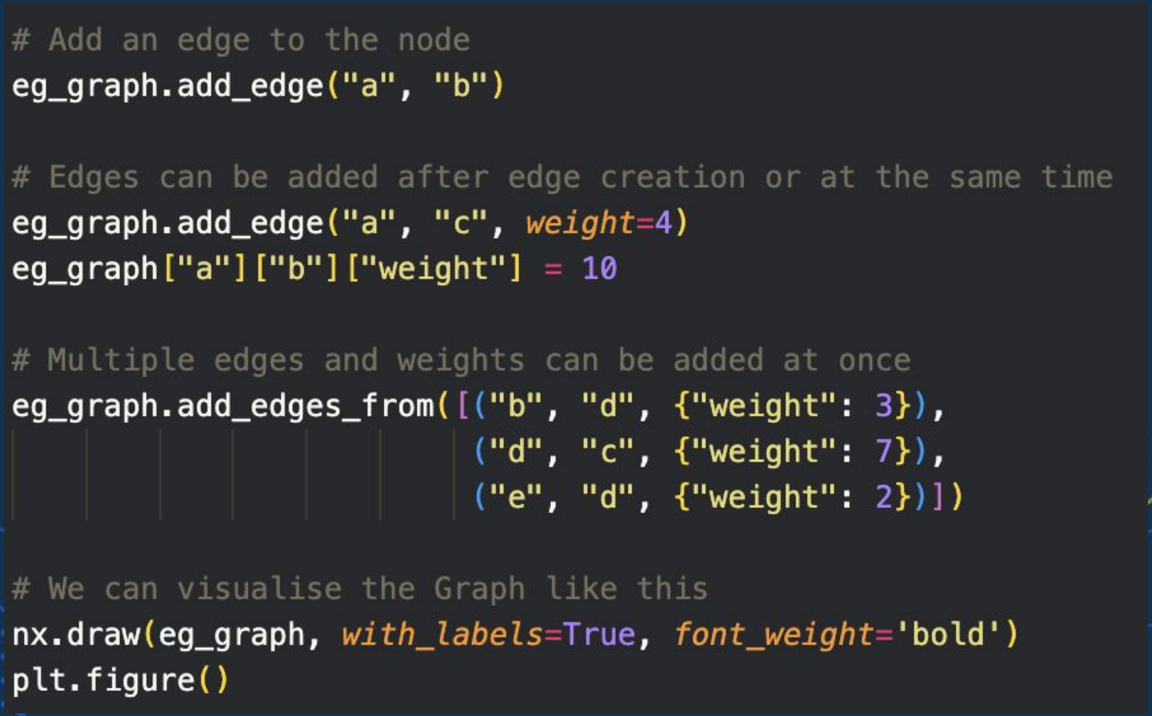
CoGrammar

- The best implementation of Graphs is by using a package called **NetworkX** or **JSNetworkX** (install using pip or npm).



```python
import networkx as nx
import matplotlib.pyplot as plt

# Create a new graph
eg_graph = nx.Graph()

# Add a node to the graph
eg_graph.add_node("a")

# Add a list of nodes to the graph
nodes = ["b", "c", "d", "e"]

# You can also add node atrributes to each node using the form:
# ("b", {colour: "blue"})
```

❖ NetworkX allows use to **add nodes, edges, weights, attributes and directions** as well as **visualise graphs**, easily and efficiently.



```
# Add an edge to the node
eg_graph.add_edge("a", "b")

# Edges can be added after edge creation or at the same time
eg_graph.add_edge("a", "c", weight=4)
eg_graph["a"]["b"]["weight"] = 10

# Multiple edges and weights can be added at once
eg_graph.add_edges_from([("b", "d", {"weight": 3}),
                         ("d", "c", {"weight": 7}),
                         ("e", "d", {"weight": 2})])

# We can visualise the Graph like this
nx.draw(eg_graph, with_labels=True, font_weight='bold')
plt.figure()
```

What is the degree of a node in a graph?

A. The number of nodes surrounding the node
B. The sum of the weights of the edges from the node
C. The number of edges connected to a node
D. The label assigned to a node

CoGrammar

What is a cycle in a graph?

A. A path that visits every node at least once

B. A path that starts and ends at the same node

C. A set of edges that has the same weight in both directions

D. A node that has no edges

Homework

Practise the skills we've developed by completing the following problems:

❖ The next slide contains two questions to test your theoretical understanding of graphs in a real world scenario.

❖ We'll be going through two LeetCode examples in the lecture over the weekend, attempt them yourself and come ready with questions:
  - ➢ Example 1
  - ➢ Example 2

❖ Practice speaking through your solutions and explaining how you approached each problem.

Homework Example

Consider a delivery company wanting to visualise their network of package collection points to help improve efficiency and organisation of delivery routes.

Start	End	Distance
A	B	10
C	D	12
A	D	7
B	C	4
B	E	8

CoGrammar

1. Using the package collection points provided, create a graph for this scenario.
   a. Determine the weights of each link.
   b. Add directions to the links.
   c. What type of graph is this?

1. What is the shortest route that visits all the collection points, assuming all links are bidirectional? (Use the minimum_spanning_tree method in Python)

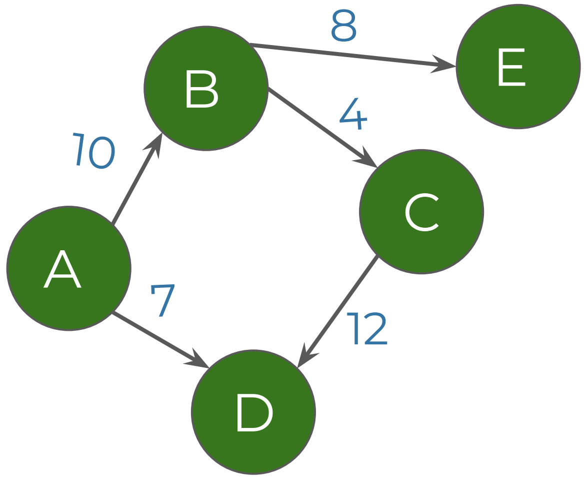
Homework Example

Consider a delivery company wanting to visualise their network of package collection points to help improve efficiency and organisation of delivery routes.

Start	End	Distance
A	B	10
C	D	12
A	D	7
B	C	4
B	E	8

CoGrammar

1. Using the package collection points provided, create a graph for this scenario.
   a. Determine the weights of each link.
   b. Add directions to the links.
   c. What type of graph is this?



Directed Acyclic Graph

Homework Example

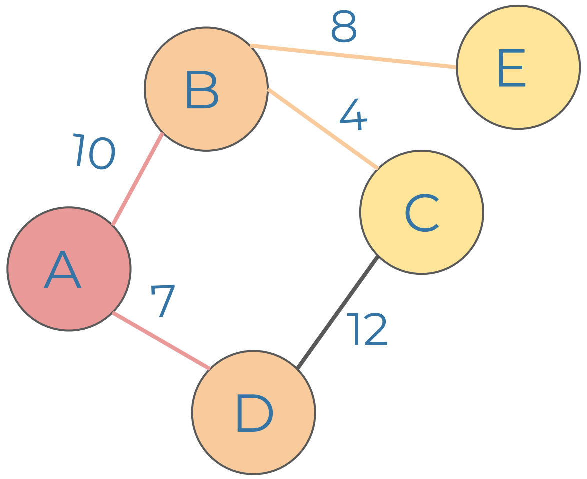
Consider a delivery company wanting to visualise their network of package collection points to help improve efficiency and organisation of delivery routes.

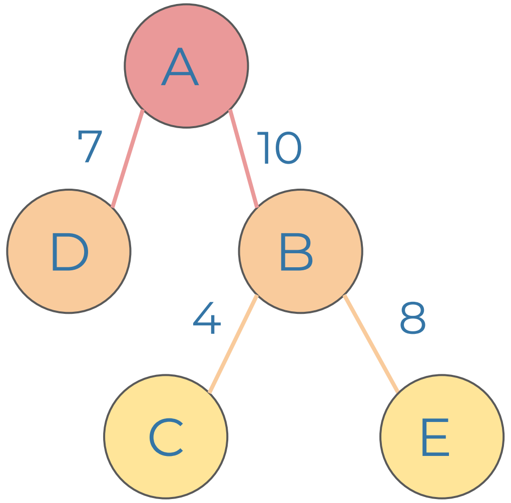
Start	End	Distance
A	B	10
C	D	12
A	D	7
B	C	4
B	E	8

CoGrammar

2. What is the shortest route that visits all the collection points, assuming all links are bidirectional? (Use the minimum_spanning_tree method in Python)

Start with A

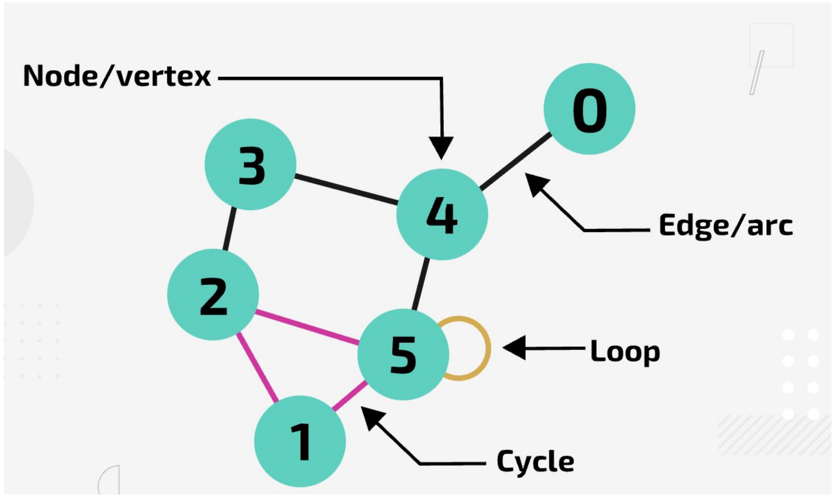




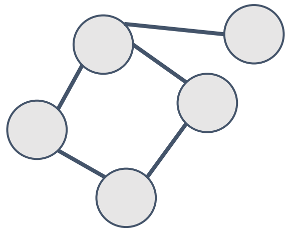
Summary

Graphs

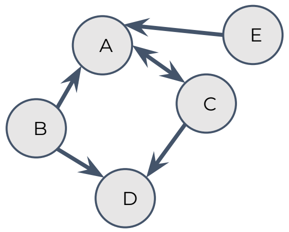
- ★ Non-linear data structure
- ★ Made up of nodes/vertices
- ★ Connected by edges/links



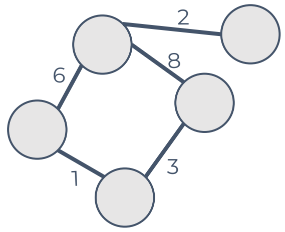
Types of Graphs



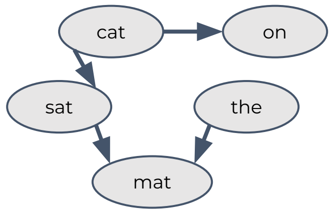
Undirected Graphs



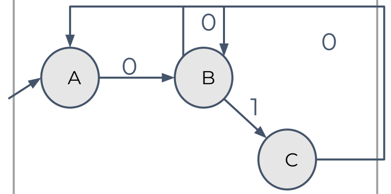
Directed Graphs



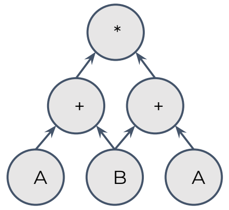
Weighted Graphs



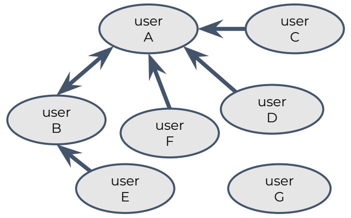
Vertex Labelled Graphs



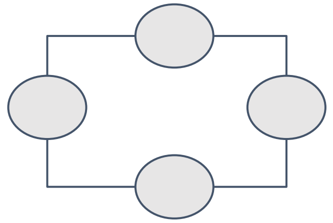
Cyclic Graphs



Directed Acyclic Graphs



Disconnected Graphs



Connected Graphs

Further Learning

- ❖ Ada Computer Science - Introduction to Graphs with lots of visuals
- ❖ Portland State University - Comprehensive guide to graphs, with a more Mathematics centered approach
- ❖ Simplilearn - In-depth introduction to graphs, covers the different types well, lots of visual guides
- ❖ GeeksForGeeks - Prim's Algorithm explanation and implementation

CoGrammar

Q & A SECTION

Please use this time to ask any questions relating to the Graphs, should you have any.

Thank you for attending

CoGrammar

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education